CS 31 Week 8 Discussion 2E

Srinath

Announcements

- Project 6 is up! Due **11:00 PM Monday, November 21**

Outline

- Pointers
- Structs & Classes
- Worksheet 7 & 8

Pointers

Pointers : Basics

What is a pointer ?

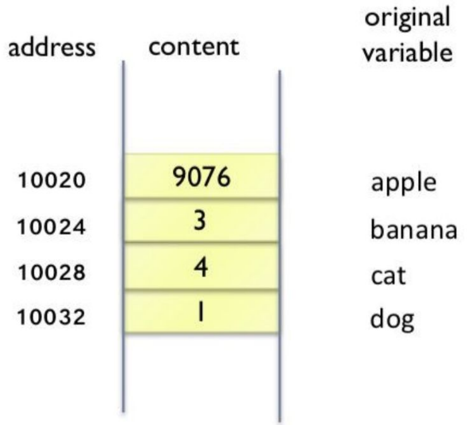
Pointers : Basics

What is a pointer ?

Pointers are symbolic representations of memory addresses.

All the variables defined in a program must be stored in memory, pointers help us store that memory address and operate on the variables indirectly.

```
int main() {
    int apple;
    int banana = 3;
    int cat = 4;
    int dog = cat - banana;

    ...

    return 0;
}
```

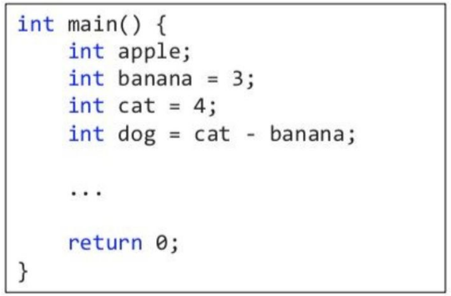


Pointers : Basics

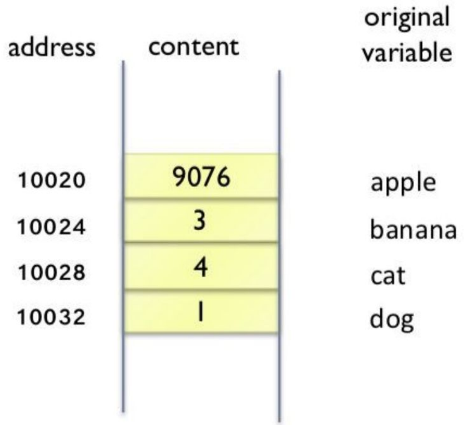
What is a pointer ?

Pointers are symbolic representations of memory addresses.

All the variables defined in a program must be stored in memory, pointers help us store that memory address and operate on the variables indirectly.



```
int main() {
    int apple;
    int banana = 3;
    int cat = 4;
    int dog = cat - banana;

    ...

    return 0;
}
```



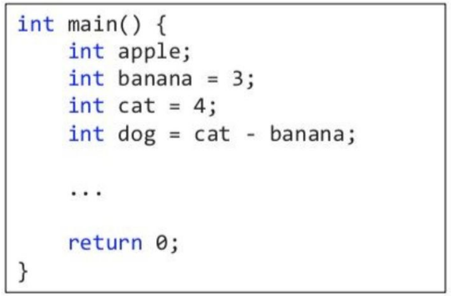
Is memory address an integer?

Pointers : Basics

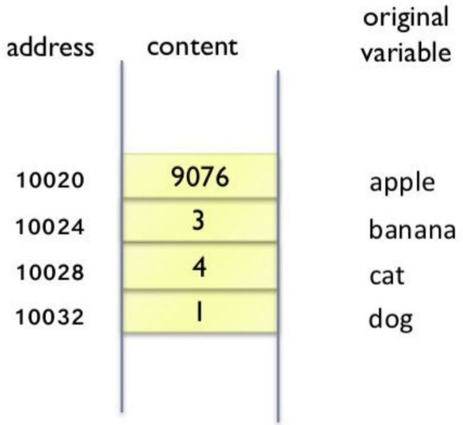
What is a pointer ?

Pointers are symbolic representations of memory addresses.

All the variables defined in a program must be stored in memory, pointers help us store that memory address and operate on the variables indirectly.



```
int main() {
    int apple;
    int banana = 3;
    int cat = 4;
    int dog = cat - banana;

    ...

    return 0;
}
```



Is memory address an integer?

- At CPU level, any pointer (or memory address) can be treated as an integer.

So is a pointer variable just same as an integer variable?

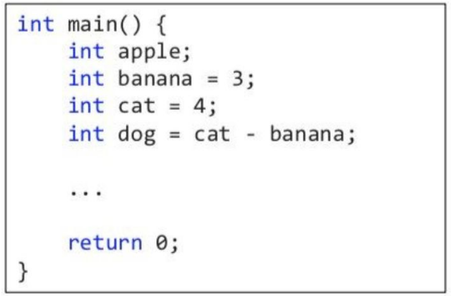
-

Pointers : Basics

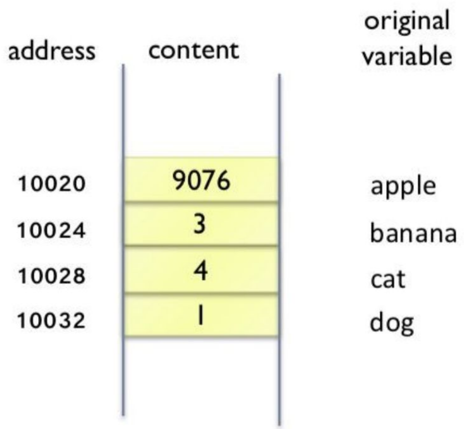
What is a pointer ?

Pointers are symbolic representations of memory addresses.

All the variables defined in a program must be stored in memory, pointers help us store that memory address and operate on the variables indirectly.



```
int main() {
    int apple;
    int banana = 3;
    int cat = 4;
    int dog = cat - banana;

    ...

    return 0;
}
```



Is memory address an integer?

- At CPU level, any pointer (or memory address) can be treated as an integer.

So is a pointer variable just same as an integer variable?

- No, using pointer you can manipulate object directly in memory, which cannot be done by variable of type int.

Pointers : Basics

Declaration

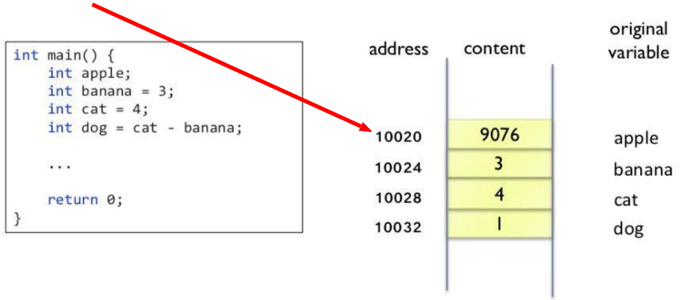
pointer_type * pointer_name ;

Examples :

*int * ip;*	*// pointer to an integer*
*double *dp;*	*// pointer to a double*
float fp;*	*// pointer to a float*
*char *cp;*	*// pointer to a character*

The compiler doesn't really care if the * is attached to variable name or type.

*int *apple_ptr = &apple;*



```
int main() {
    int apple;
    int banana = 3;
    int cat = 4;
    int dog = cat - banana;

    ...

    return 0;
}
```

Pointers : Basics

Declaration

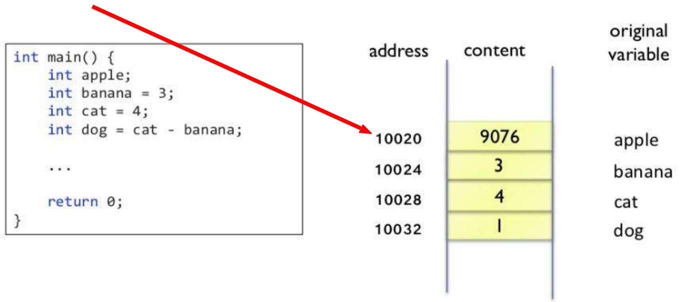
pointer_type * pointer_name ;

Examples :

*int * ip;*	*// pointer to an integer*
*double *dp;*	*// pointer to a double*
float fp;*	*// pointer to a float*
*char *cp;*	*// pointer to a character*

The compiler doesn't really care if the * is attached to variable name or type.

*int *apple_ptr = &apple;*



```
int main() {
    int apple;
    int banana = 3;
    int cat = 4;
    int dog = cat - banana;

    ...

    return 0;
}
```

address	content	original variable
10020	9076	apple
10024	3	banana
10028	4	cat
10032	1	dog

Can this be done instead?

*int *apple_ptr = (int *) 10020;*

Pointers : Basics

Declaration

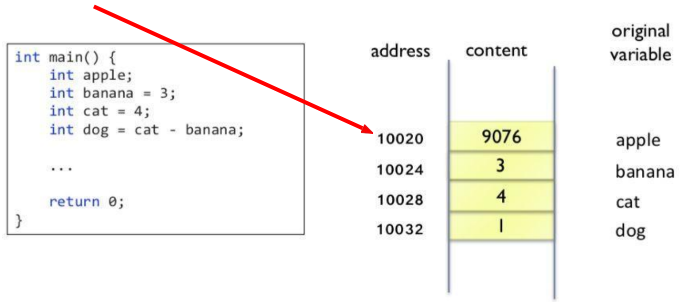
pointer_type * pointer_name ;

Examples :

*int * ip;*	*// pointer to an integer*
*double *dp;*	*// pointer to a double*
float fp;*	*// pointer to a float*
*char *cp;*	*// pointer to a character*

The compiler doesn't really care if the * is attached to variable name or type.

*int *apple_ptr = &apple;*



```
int main() {
    int apple;
    int banana = 3;
    int cat = 4;
    int dog = cat - banana;

    ...

    return 0;
}
```

Can this be done instead?

*int *apple_ptr = (int *) 10020;*

- Yes, but the address assigned to the variable apple by compiler is not known to us.

Pointers : operators *, &

***** is **dereference** operator

- *int * a;*   // a is a pointer variable.
- **a* means to retrieve the value at address stored in a

& is **address-of** operator

- int v;   // v is an integer variable.
- &v means to get the memory address of variable v

Pointers : operators *, &

* is **dereference** operator

- *int * a;* // a is a pointer variable.
- **a* means to retrieve the value at address stored in a

& is **address-of** operator

- int v; // v is an integer variable.
- &v means to get the memory address of variable v

```
int v =10;
int * a = &v;

cout << v << " , " << *a << endl;                    // output? -
```

Pointers : operators *, &

* is **dereference** operator

- *int * a;* // a is a pointer variable.
- **a* means to retrieve the value at address stored in a

& is **address-of** operator

- int v; // v is an integer variable.
- &v means to get the memory address of variable v

```
int v =10;
int * a = &v;

cout << v << " , " << *a << endl;                    // output? - 10 , 10

*a = 90;
cout << v << " , " << *a << endl;                    // output? -
```

Pointers : operators *, &

* is **dereference** operator
- *int * a;* // a is a pointer variable.
- **a* means to retrieve the value at address stored in a

& is **address-of** operator
- int v; // v is an integer variable.
- &v means to get the memory address of variable v

```
int v =10;
int * a = &v;

cout << v << “ , ” << *a << endl;          // output? - 10 , 10

*a = 90;
cout << v << “ , ” << *a << endl;          // output? - 90 , 90

v = 80
cout << v << “ , ” << *a << endl;          // output? -
```

Pointers : operators *, &

* is **dereference** operator
- *int * a;* // a is a pointer variable.
- **a* means to retrieve the value at address stored in a

& is **address-of** operator
- int v; // v is an integer variable.
- &v means to get the memory address of variable v

```
int v =10;
int * a = &v;

cout << v << " , " << *a << endl;          // output? - 10 , 10

*a = 90;
cout << v << " , " << *a << endl;          // output? - 90 , 90

v = 80
cout << v << " , " << *a << endl;          // output? - 80 , 80

cout << a << endl;                         // output? -
```

Pointers : operators *, &

* is **dereference** operator

- *int * a;* // a is a pointer variable.
- **a* means to retrieve the value at address stored in a

& is **address-of** operator

- int v; // v is an integer variable.
- &v means to get the memory address of variable v

```
int v =10;
int * a = &v;

cout << v << “ , ” << *a << endl;              // output? - 10 , 10

*a = 90;
cout << v << “ , ” << *a << endl;              // output? - 90 , 90

v = 80
cout << v << “ , ” << *a << endl;              // output? - 80 , 80

cout << a << endl;                             // output? - memory address say 0x77ffedc.

cout << &v << endl;                            // output? -
```

Pointers : operators *, &

* is **dereference** operator
- *int * a;* // a is a pointer variable.
- **a* means to retrieve the value at address stored in a

& is **address-of** operator
- int v; // v is an integer variable.
- &v means to get the memory address of variable v

```
int v =10;
int * a = &v;

cout << v << “ , ” << *a << endl;          // output? - 10 , 10

*a = 90;
cout << v << “ , ” << *a << endl;          // output? - 90 , 90

v = 80
cout << v << “ , ” << *a << endl;          // output? - 80 , 80

cout << a << endl;                         // output? - memory address say 0x77ffedc.

cout << &v << endl;                        // output? - same memory address as above
```

What if we try doing *v ?
-

Pointers : operators *, &

* is **dereference** operator

- *int * a;* // a is a pointer variable.
- **a* means to retrieve the value at address stored in a

& is **address-of** operator

- int v; // v is an integer variable.
- &v means to get the memory address of variable v

```
int v =10;
int * a = &v;

cout << v << “ , ” << *a << endl;          // output? - 10 , 10

*a = 90;
cout << v << “ , ” << *a << endl;          // output? - 90 , 90

v = 80
cout << v << “ , ” << *a << endl;          // output? - 80 , 80

cout << a << endl;                         // output? - memory address say 0x77ffedc.

cout << &v << endl;                        // output? - same memory address as above
```

What if we try doing *v ?

- Cannot dereference int.

How about &a ?

Pointers : operators *, &

***** is **dereference** operator

- *int * a;* // a is a pointer variable.
- **a* means to retrieve the value at address stored in a

& is **address-of** operator

- int v; // v is an integer variable.
- &v means to get the memory address of variable v

```
int v =10;
int * a = &v;

cout << v << " , " << *a << endl;          // output? - 10 , 10

*a = 90;
cout << v << " , " << *a << endl;          // output? - 90 , 90

v = 80
cout << v << " , " << *a << endl;          // output? - 80 , 80

cout << a << endl;                         // output? - memory address say 0x77ffedc.

cout << &v << endl;                        // output? - same memory address as above
```

What if we try doing *v ?

- Cannot dereference int.

How about &a ?

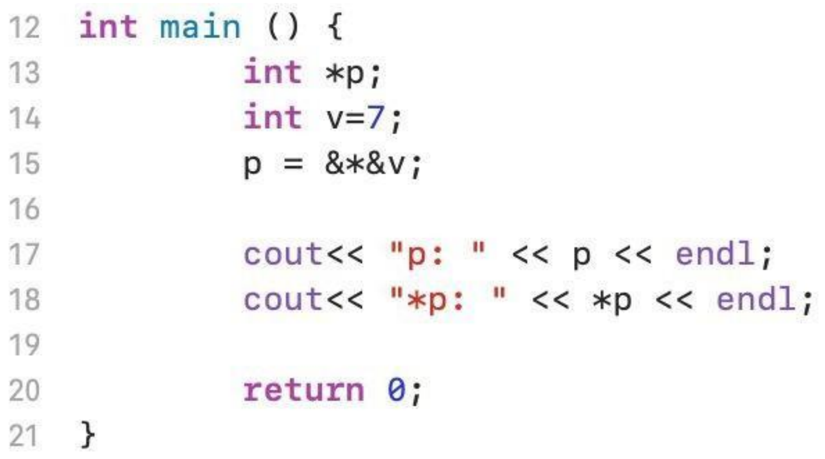
- Even pointer variable has to be stored in memory
- It grabs the address of pointer variable.
- So, it is pointer to a pointer!!

Pointers : operators *, &

```
int main () {
        int *p;
        int v=7;
        p = &*&v;

        cout<< "p: " << p << endl;
        cout<< "*p: " << *p << endl;

        return 0;
}
```

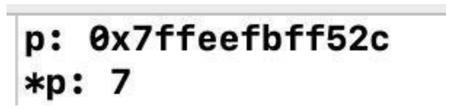
Output : ?

Pointers : operators *, &



```
int main () {
        int *p;
        int v=7;
        p = &*&v;

        cout<< "p: " << p << endl;
        cout<< "*p: " << *p << endl;

        return 0;
}
```

Output : ?

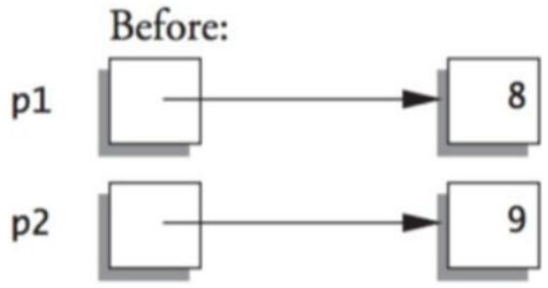


```
p: 0x7ffeefbff52c
*p: 7
```

Pointers : operator =

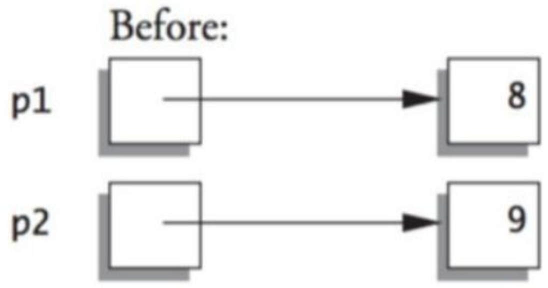
p1 = p2

Say p1, p2 are two integer pointers

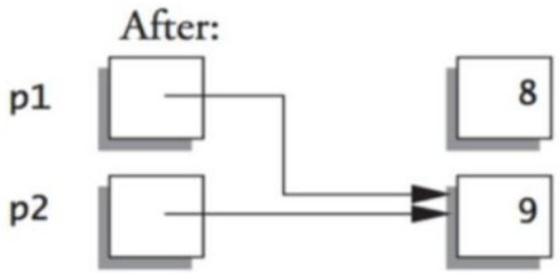


Pointers : operator =

Say p1, p2 are two integer pointers



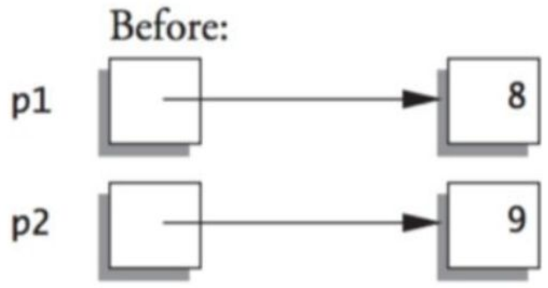
p1 = p2



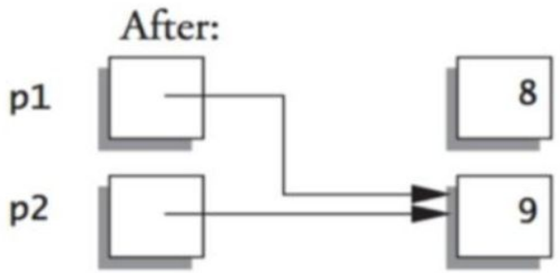
***p1 = *p2**

Pointers : operator =

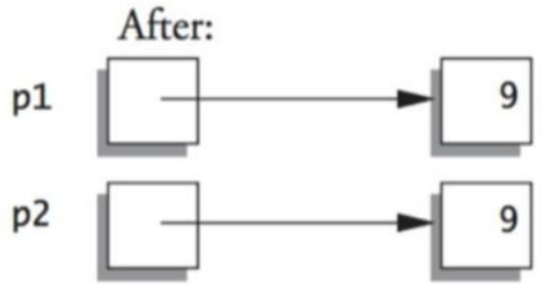
Say p1, p2 are two integer pointers



p1 = p2



*p1 = *p2



Pointers : with Arrays

The name of the array is simply a pointer.
Pointers can be used to point to an item in an array.

```
int grades[100];
int * p = grades;
```

p or **grades** point to the first element of the array.

Pointers : with Arrays

The name of the array is simply a pointer.
Pointers can be used to point to an item in an array.

```
int grades[100];
int * p = grades;
```

p or **grades** point to the first element of the array.

```
// Traversing array using pointer

double grades[5];
for(double* p = grades; p < grades + 5;  p++) {
            *p = 65.0;
}
```

Pointers : with Arrays

The name of the array is simply a pointer.
Pointers can be used to point to an item in an array.

```
int grades[100];
int * p = grades;
```

p or **grades** point to the first element of the array.

```
// Traversing array using pointer

double grades[5];
for(double* p = grades; p < grades + 5;  p++) {
        *p = 65.0;
}
```

What will be the contents of array **grades** after the loop?

Pointers : with Arrays

The name of the array is simply a pointer.
Pointers can be used to point to an item in an array.

```
int grades[100];
int * p = grades;
```

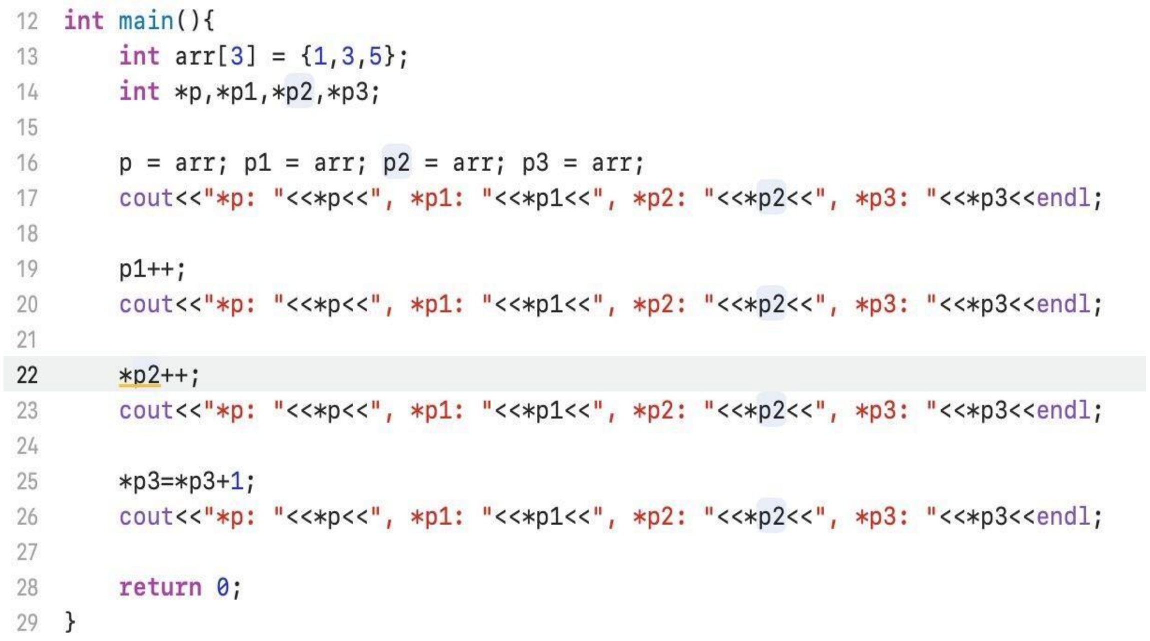
p or **grades** point to the first element of the array.

```
// Traversing array using pointer

double grades[5];
for(double* p = grades; p < grades + 5;  p++) {
        *p = 65.0;
}
```

What will be the contents of array **grades** after the loop?

- All values set to 65.0

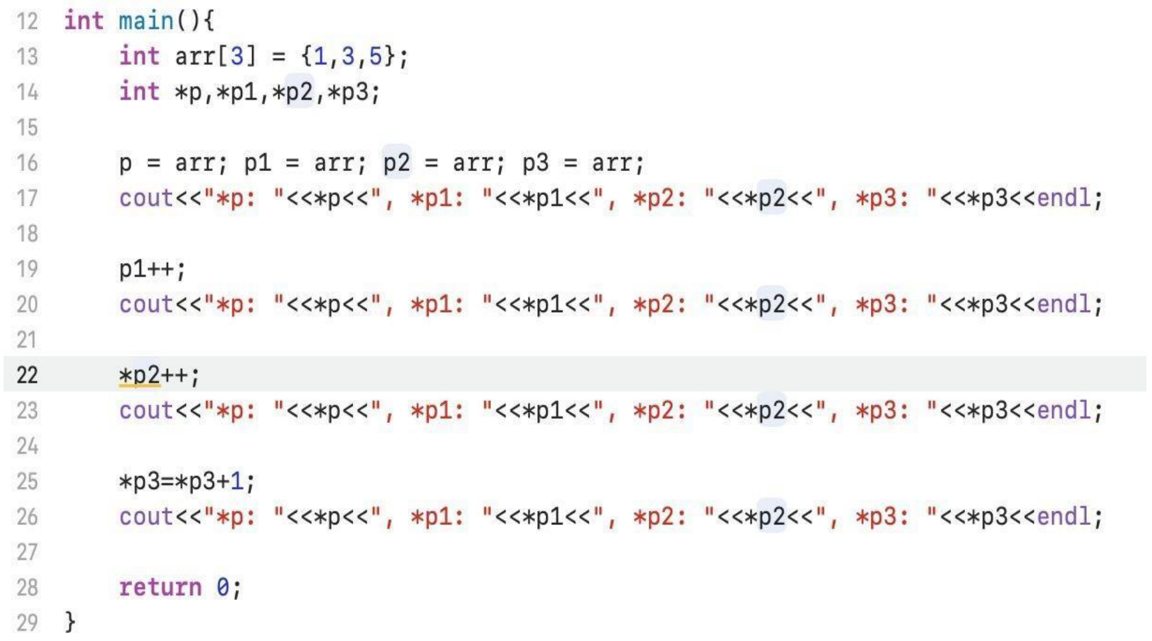
Pointers : with Arrays



```cpp
int main(){
    int arr[3] = {1,3,5};
    int *p,*p1,*p2,*p3;

    p = arr; p1 = arr; p2 = arr; p3 = arr;
    cout<<"*p: "<<*p<<", *p1: "<<*p1<<", *p2: "<<*p2<<", *p3: "<<*p3<<endl;

    p1++;
    cout<<"*p: "<<*p<<", *p1: "<<*p1<<", *p2: "<<*p2<<", *p3: "<<*p3<<endl;

    *p2++;
    cout<<"*p: "<<*p<<", *p1: "<<*p1<<", *p2: "<<*p2<<", *p3: "<<*p3<<endl;

    *p3=*p3+1;
    cout<<"*p: "<<*p<<", *p1: "<<*p1<<", *p2: "<<*p2<<", *p3: "<<*p3<<endl;

    return 0;
}
```

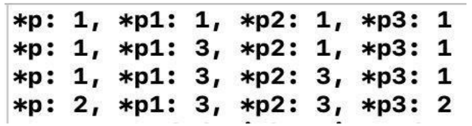
Output : ?

Pointers : with Arrays



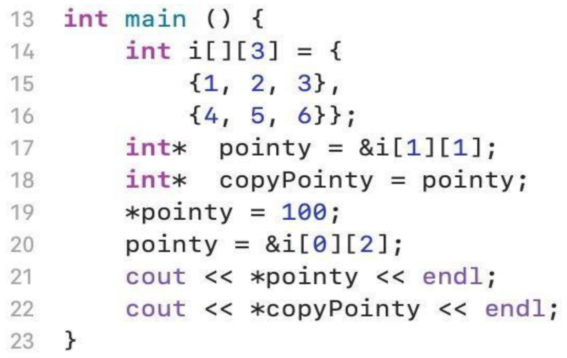
```
int main(){
    int arr[3] = {1,3,5};
    int *p,*p1,*p2,*p3;

    p = arr; p1 = arr; p2 = arr; p3 = arr;
    cout<<"*p: "<<*p<<", *p1: "<<*p1<<", *p2: "<<*p2<<", *p3: "<<*p3<<endl;

    p1++;
    cout<<"*p: "<<*p<<", *p1: "<<*p1<<", *p2: "<<*p2<<", *p3: "<<*p3<<endl;

    *p2++;
    cout<<"*p: "<<*p<<", *p1: "<<*p1<<", *p2: "<<*p2<<", *p3: "<<*p3<<endl;

    *p3=*p3+1;
    cout<<"*p: "<<*p<<", *p1: "<<*p1<<", *p2: "<<*p2<<", *p3: "<<*p3<<endl;

    return 0;
}
```

Output : ?



```
*p: 1, *p1: 1, *p2: 1, *p3: 1
*p: 1, *p1: 3, *p2: 1, *p3: 1
*p: 1, *p1: 3, *p2: 3, *p3: 1
*p: 2, *p1: 3, *p2: 3, *p3: 2
```

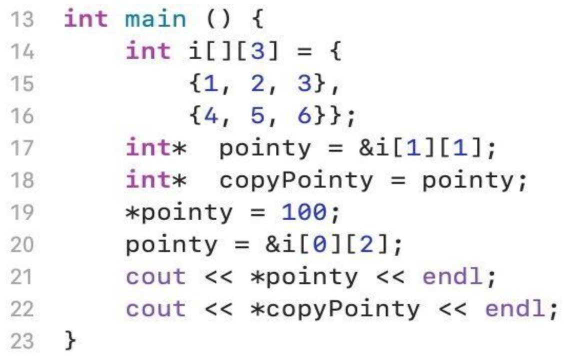
Pointers : with Arrays



```
int main () {
    int i[][3] = {
        {1, 2, 3},
        {4, 5, 6}};
    int*  pointy = &i[1][1];
    int*  copyPointy = pointy;
    *pointy = 100;
    pointy = &i[0][2];
    cout << *pointy << endl;
    cout << *copyPointy << endl;
}
```

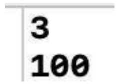
Output : ?

Pointers : with Arrays

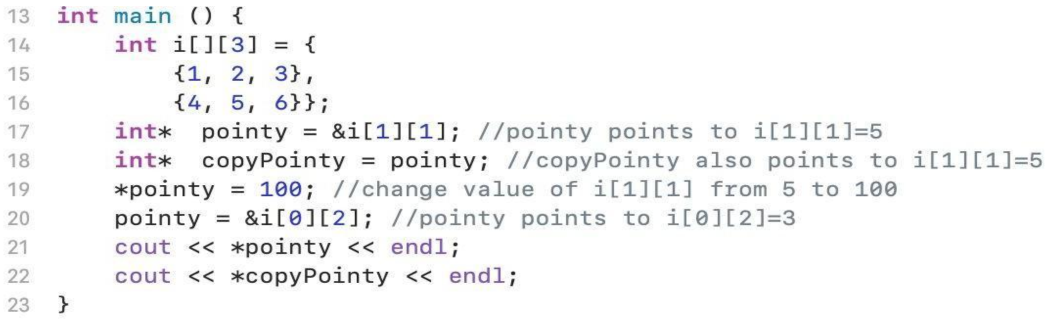


```cpp
int main () {
    int i[][3] = {
        {1, 2, 3},
        {4, 5, 6}};
    int*  pointy = &i[1][1];
    int*  copyPointy = pointy;
    *pointy = 100;
    pointy = &i[0][2];
    cout << *pointy << endl;
    cout << *copyPointy << endl;
}
```

Output : ?



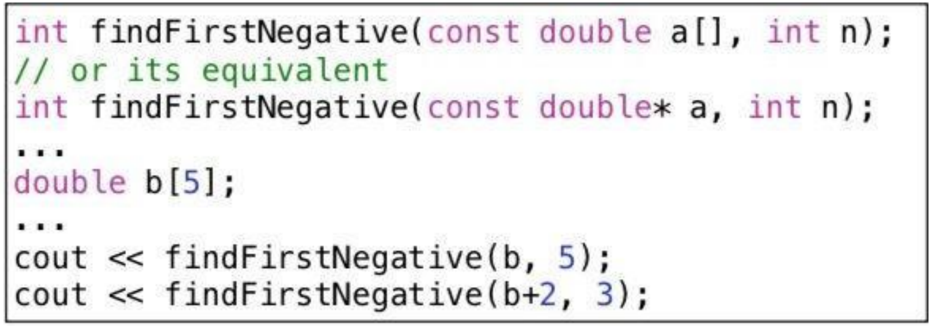
```
3
100
```



```cpp
int main () {
    int i[][3] = {
        {1, 2, 3},
        {4, 5, 6}};
    int*  pointy = &i[1][1]; //pointy points to i[1][1]=5
    int*  copyPointy = pointy; //copyPointy also points to i[1][1]=5
    *pointy = 100; //change value of i[1][1] from 5 to 100
    pointy = &i[0][2]; //pointy points to i[0][2]=3
    cout << *pointy << endl;
    cout << *copyPointy << endl;
}
```

Pointers : with Arrays

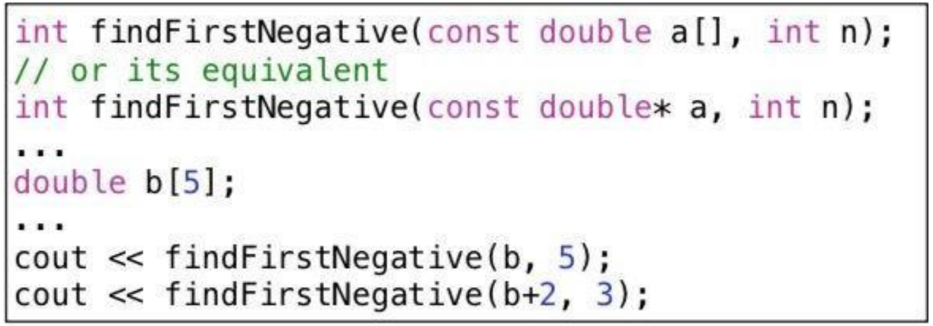
Pointers can be used to pass arrays to a function



```
int findFirstNegative(const double a[], int n);
// or its equivalent
int findFirstNegative(const double* a, int n);
...
double b[5];
...
cout << findFirstNegative(b, 5);
cout << findFirstNegative(b+2, 3);
```

Pointers : with Arrays

Pointers can be used to pass arrays to a function



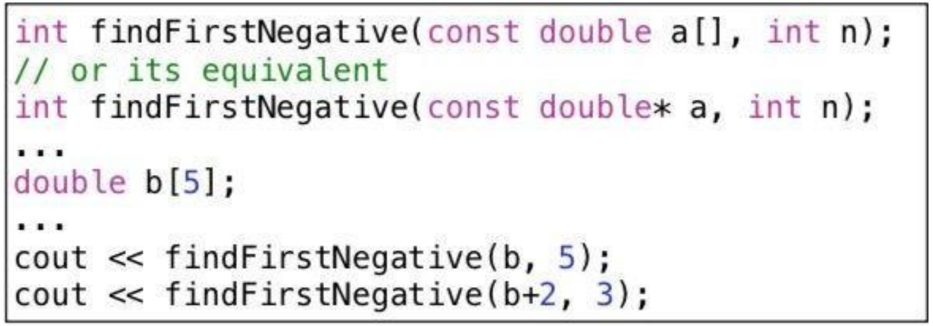
```
int findFirstNegative(const double a[], int n);
// or its equivalent
int findFirstNegative(const double* a, int n);
...
double b[5];
...
cout << findFirstNegative(b, 5);
cout << findFirstNegative(b+2, 3);
```

int grades[10];
int * p;
grades = p ;

Can this be done?

Pointers : with Arrays

Pointers can be used to pass arrays to a function



```
int findFirstNegative(const double a[], int n);
// or its equivalent
int findFirstNegative(const double* a, int n);
...
double b[5];
...
cout << findFirstNegative(b, 5);
cout << findFirstNegative(b+2, 3);
```

int grades[10];
int * p;
grades = p ;

Can this be done?

- No. You cannot change the pointer value in an array variable

Pointers : with Functions

```
void my_function(int * temp) {
        *temp = 50;
        cout << “Inside” << *temp << endl;
}

int main() {

        int a = 100;
        int * p = &a;
        cout << “Before” << *p << endl;
        my_function(p);
        cout << “After” << *p << endl;
        return 0;
}
```

Output : ?

Pointers : with Functions

```
void my_function(int * temp) {
        *temp = 50;
        cout << “Inside” << *temp << endl;
}

int main() {

        int a = 100;
        int * p = &a;
        cout << “Before” << *p << endl;
        my_function(p);
        cout << “After” << *p << endl;
        return 0;
}
```

Output : ?

```
-    Before 100
     Inside 50
     After 50
```

Pointers : Axioms

Some axioms(true by default) on pointers and pointer arrays

*&x ⇒ x

&a[i] + j ⇒ ?

Pointers : Axioms

Some axioms(true by default) on pointers and pointer arrays

$*\&x \Rightarrow x$

$\&a[i] + j \Rightarrow \&a[i+j]$

$\&a[i] - j \Rightarrow ?$

Pointers : Axioms

Some axioms(true by default) on pointers and pointer arrays

$$*\&x \Rightarrow x$$

$$\&a[i] + j \Rightarrow \&a[i+j]$$

$$\&a[i] - j \Rightarrow \&a[i-j]$$

$$\&a[i] < \&a[j] \Rightarrow ?$$

Pointers : Axioms

Some axioms(true by default) on pointers and pointer arrays

$*\&x \Rightarrow x$

$\&a[i] + j \Rightarrow \&a[i+j]$

$\&a[i] - j \Rightarrow \&a[i-j]$

$\&a[i] < \&a[j] \Rightarrow i < j$ (also for <=, >, >=, ==, !=)

$\&a[i] - \&a[j] \Rightarrow$?

Pointers : Axioms

Some axioms(true by default) on pointers and pointer arrays

*&x ⇒ x

&a[i] + j ⇒ &a[i+j]

&a[i] - j ⇒ &a[i-j]

&a[i] < &a[j] ⇒ i < j (also for <=, >, >=, ==, !=)

&a[i] - &a[j] ⇒ i - j

p[i] <==> ?

Pointers : Axioms

Some axioms(true by default) on pointers and pointer arrays

*&x ⇒ x

&a[i] + j ⇒ &a[i+j]

&a[i] - j ⇒ &a[i-j]

&a[i] < &a[j] ⇒ i < j (also for <=, >, >=, ==, !=)

&a[i] - &a[j] ⇒ i - j

p[i] <==> *(p+i)

Structs/Classes

Struct : Basics

Struct is a Composite data type.
A collection of values of different types or other structs! (member variables)

// Defining a struct

```
struct <structName> {
    <member1_type> <member1_name>;
    <member2_type> <member2_name>;
    // ...etc.
}; // Remember the semicolon!
```

// Declaring a struct
structName v1, v2;

```
struct Employee {
        string name;
        double salary;
        int age;
};

int main() {
        Employee emp1, emp2;
}
```

Struct : Basics

Access struct member variables ?

```
struct Employee {
        string name;
        double salary;
        int age;
};

int main() {
        Employee emp1, emp2;
}
```

Struct : Basics

Access struct member variables ?

- Use dot(.) operator

```
Employee emp;
cout << emp.name << endl;
cout << emp.salary << endl;
```

Initialization ?

```
struct Employee {
        string name;
        double salary;
        int age;
};

int main() {
        Employee emp1, emp2;
}
```

Struct : Basics

Access struct member variables ?

- Use dot(.) operator

```
Employee emp;
cout << emp.name << endl;
cout << emp.salary << endl;
```

Initialization ?

```
Employee emp;
emp.name = “Jade”;
emp.salary = 0.0;
emp.age = 19;
```

```
struct Employee {
        string name;
        double salary;
        int age;
};

int main() {
        Employee emp1, emp2;
}
```

Struct : Basics

Access struct member variables ?

- Use dot(.) operator

```
Employee emp;
cout << emp.name << endl;
cout << emp.salary << endl;
```

Initialization ?

```
Employee emp;
emp.name = “Jade”;
emp.salary = 0.0;
emp.age = 19;
```

Alternatively

```
Employee emp = {“jade”, 0.0, 19};
```

```
struct Employee {
          string name;
          double salary;
          int age;
};

int main() {
          Employee emp1, emp2;
}
```

Struct : Basics

Access struct member variables ?

- Use dot(.) operator

```
Employee emp;
cout << emp.name << endl;
cout << emp.salary << endl;
```

Initialization ?

```
Employee emp;
emp.name = “Jade”;
emp.salary = 0.0;
emp.age = 19;
```

Alternatively

```
Employee emp = {“jade”, 0.0, 19};

Employee emp = {“jade”, 0.0, 19, “hii”};  // will this compile?
```

```
struct Employee {
        string name;
        double salary;
        int age;
};

int main() {
        Employee emp1, emp2;
}
```

Struct : Basics

Access struct member variables ?

- Use dot(.) operator

```
Employee emp;
cout << emp.name << endl;
cout << emp.salary << endl;
```

Initialization ?

```
Employee emp;
emp.name = “Jade”;
emp.salary = 0.0;
emp.age = 19;
```

Alternatively

```
Employee emp = {“jade”, 0.0, 19};

Employee emp = {“jade”, 0.0, 19, “hii”};  // will this compile? - No

Employee emp = {“jade”, 0.0}; // will this compile? -
```

```
struct Employee {
        string name;
        double salary;
        int age;
};

int main() {
        Employee emp1, emp2;
}
```

Struct : Basics

```
struct Employee {
        string name;
        double salary;
        int age;
};

int main() {
        Employee emp1, emp2;
}
```

Access struct member variables ?

- Use dot(.) operator

```
Employee emp;
cout << emp.name << endl;
cout << emp.salary << endl;
```

Initialization ?

```
Employee emp;
emp.name = “Jade”;
emp.salary = 0.0;
emp.age = 19;
```

Alternatively

```
Employee emp = {“jade”, 0.0, 19};

Employee emp = {“jade”, 0.0, 19, “hii”};  // will this compile? - No

Employee emp = {“jade”, 0.0}; // will this compile? - Yes
```

- Use exactly the same order
- More values => error
- Less values => assign to the members in order

Struct : Basics

Access struct member variables ?

- Use dot(.) operator

```
Employee emp;
cout << emp.name << endl;
cout << emp.salary << endl;
```

Initialization ?

```
Employee emp;
emp.name = “Jade”;
emp.salary = 0.0;
emp.age = 19;
```

Alternatively

```
Employee emp = {“jade”, 0.0, 19};

Employee emp = {“jade”, 0.0, 19, “hii”};  // will this compile? - No

Employee emp = {“jade”, 0.0}; // will this compile? - Yes
```

```
struct Employee {
        string name;
        double salary;
        int age;
};

int main() {
        Employee emp1, emp2;
}
```

- Use exactly the same order
- More values => error
- Less values => assign to the members in order

```
// will this compile? -
Employee emp;
emp = {“jade”, 0.0};
```

Struct : Basics

```
struct Employee {
        string name;
        double salary;
        int age;
};

int main() {
        Employee emp1, emp2;
}
```

Access struct member variables ?

- Use dot(.) operator

```
Employee emp;
cout << emp.name << endl;
cout << emp.salary << endl;
```

Initialization ?

```
Employee emp;
emp.name = “Jade”;
emp.salary = 0.0;
emp.age = 19;
```

Alternatively

```
Employee emp = {“jade”, 0.0, 19};

Employee emp = {“jade”, 0.0, 19, “hii”};  // will this compile? - No

Employee emp = {“jade”, 0.0}; // will this compile? - Yes
```

- Use exactly the same order
- More values => error
- Less values => assign to the members in order

```
// will this compile? - No
Employee emp;
emp = {“jade”, 0.0};
```

- Can only use it with declaration and if all members are public.

Struct : Pointer to struct

A struct is also stored in memory, so has some address too!

```
Employee emp;
Employee * emp_ptr;
```

How to get address of struct?

-

```
struct Employee {
        string name;
        double salary;
        int age;
};

int main() {
        Employee emp1, emp2;
}
```

Struct : Pointer to struct

A struct is also stored in memory, so has some address too!

```
Employee emp;
Employee * emp_ptr;
```

How to get address of struct?

- The &(address-of) operator

```
emp_ptr = &emp;
```

Access member variables using pointer?

```
struct Employee {
        string name;
        double salary;
        int age;
};

int main() {
        Employee emp1, emp2;
}
```

Struct : Pointer to struct

A struct is also stored in memory, so has some address too!

```
Employee emp;
Employee * emp_ptr;
```

How to get address of struct?

- The &(address-of) operator

```
emp_ptr = &emp;
```

Access member variables using pointer?

```
cout << (*emp_ptr).name << endl;
cout << (*emp_ptr).salary << endl;
```

Or

```
cout << emp_ptr -> name << endl; // notice the '->' operator.
```

```
struct Employee {
        string name;
        double salary;
        int age;
};

int main() {
        Employee emp1, emp2;
}
```

'->' only works for pointer!

Struct : Public/Private

Public/Private give us access control to variables and functions

- Struct is ‘public’ by default
- Class is ‘private’ by default

Private variables or members cannot be accessed directly.

```
struct Employee {
                string name;
                double salary;
        private:
                int age;
};

int main() {
        Employee emp1, emp2;
}
```

Struct : Public/Private

Public/Private give us access control to variables and functions
- Struct is ‘public’ by default
- Class is ‘private’ by default

Private variables or members cannot be accessed directly.

```
struct Employee {
        string name;
        double salary;
    private:
        int age;
};

int main() {
    Employee emp1, emp2;
}
```

```
Employee emp = {“Jade”, 0.0, 19};
Employee * p = &emp;
```

Can we do this?
-

Struct : Public/Private

Public/Private give us access control to variables and functions

- Struct is ‘public’ by default
- Class is ‘private’ by default

Private variables or members cannot be accessed directly.

```
struct Employee {
                string name;
                double salary;
        private:
                int age;
};

int main() {
        Employee emp1, emp2;
}
```

```
Employee emp = {“Jade”, 0.0, 19};
Employee * p = &emp;
```

Can we do this?
- No, unless all members are public.

```
cout << emp.salary << endl;        // will this compile? -

cout << emp.age << endl;           // will this compile? -

cout << p->age << endl;            // will this compile? -

cout << p->salary << endl;         // will this compile? -
```

Struct : Public/Private

Public/Private give us access control to variables and functions

- Struct is ‘public’ by default
- Class is ‘private’ by default

Private variables or members cannot be accessed directly.

```
struct Employee {
            string name;
            double salary;
      private:
            int age;
};

int main() {
      Employee emp1, emp2;
}
```

```
Employee emp = {“Jade”, 0.0, 19};
Employee * p = &emp;
```

Can we do this?
- No, unless all members are public.

```
cout << emp.salary << endl;       // will this compile? - Yes

cout << emp.age << endl;          // will this compile? - No

cout << p->age << endl;           // will this compile? - No

cout << p->salary << endl;        // will this compile? - Yes
```

Struct : Public/Private

```
Employee emp;
Employee * p = &emp;


cout << emp.salary << endl;     // will this compile? -

cout << emp.age << endl;        // will this compile? -

cout << p->age << endl;         // will this compile? -

cout << p->salary << endl;      // will this compile? -
```

```
class Employee {
                string name;
                double salary;
        public:
                int age;
};

int main() {
        Employee emp1, emp2;
}
```

Struct : Public/Private

```
Employee emp;
Employee * p = &emp;


cout << emp.salary << endl;        // will this compile? - No

cout << emp.age << endl;           // will this compile? - Yes

cout << p->age << endl;            // will this compile? - Yes

cout << p->salary << endl;         // will this compile? - No
```

```
class Employee {
                string name;
                double salary;
        public:
                int age;
};

int main() {
        Employee emp1, emp2;
}
```

Struct :

So, private variables cannot be accessed

Ho do we modify them?

```
struct Employee {
          string m_name;
          double m_salary;
     private:
          int m_age;
};
```

Struct : Member Functions

So, private variables cannot be accessed

Ho do we modify them?

- Member functions

Like member variables, we can also define member functions in a struct/class. These functions will have access to all the member variables and hence can modify them.

How do we define member functions?

```
struct Employee {
                string m_name;
                double m_salary;
        private:
                int m_age;
};
```

Struct : Member Functions

So, private variables cannot be accessed
Ho do we modify them?

- Member functions

Like member variables, we can also define member functions in a struct/class. These functions will have access to all the member variables and hence can modify them.

How do we define member functions?
You can declare member function just like any other function (return type, name, parameters etc.)

How to invoke member functions?

```
struct Employee {
            string m_name;
            double m_salary;
      private:
            int m_age;

      public:
      void setAge(int age){
            m_age = age;
      }

      private:
      int getAge(){
            return m_age;
      }
};
```

Struct : Member Functions

So, private variables cannot be accessed
Ho do we modify them?

- Member functions

Like member variables, we can also define member functions in a struct/class. These functions will have access to all the member variables and hence can modify them.

How do we define member functions?
You can declare member function just like any other function (return type, name, parameters etc.)

How to invoke member functions?

- Use dot(.) operator
- Or '->' if using pointer of struct/classes.

```
Employee emp;
Employee * p = &emp;
emp.setAge(10);
p->setAge(11);
```

```
struct Employee {
            string m_name;
            double m_salary;
      private:
            int m_age;

      public:
      void setAge(int age){
            m_age = age;
      }

      private:
      int getAge(){
            return m_age;
      }
};
```

Struct : Member Functions

```
Employee emp;
Employee * p = &emp;

emp.setAge(19);                     // will this compile? -

cout << emp.getAge() <<endl;        // will this compile? -

cout << p->getAge() << endl;        // will this compile? -
```

```
struct Employee {
            string m_name;
            double m_salary;
      private:
            int m_age;

      public:
      void setAge(int age){
            m_age = age;
      }

      private:
      int getAge(){
            return m_age;
      }
};
```

Struct : Member Functions

```
Employee emp;
Employee * p = &emp;

emp.setAge(19);                  // will this compile? - Yes

cout << emp.getAge() <<endl;     // will this compile? - No, a private member

cout << p->getAge() << endl;     // will this compile? - No, a private member
```

```
struct Employee {
            string m_name;
            double m_salary;
      private:
            int m_age;

      public:
      void setAge(int age){
            m_age = age;
      }

      private:
      int getAge(){
            return m_age;
      }
};
```

Struct : Member Functions

```
Employee emp;
Employee * p = &emp;

emp.setAge(19);                     // will this compile? - Yes

cout << emp.getAge() <<endl;        // will this compile? - No, a private member

cout << p->getAge() << endl;        // will this compile? - No, a private member
```

Similar to private member variables, private member functions cannot be accessed directly.

Why do we need private member functions then?

```
struct Employee {
            string m_name;
            double m_salary;
      private:
            int m_age;

      public:
      void setAge(int age){
            m_age = age;
      }

      private:
      int getAge(){
            return m_age;
      }
};
```

Struct : Member Functions

```
Employee emp;
Employee * p = &emp;

emp.setAge(19);                // will this compile? - Yes

cout << emp.getAge() <<endl;   // will this compile? - No, a private member

cout << p->getAge() << endl;   // will this compile? - No, a private member
```

Similar to private member variables, private member functions cannot be accessed directly.

Why do we need private member functions then?

- Functions can get complicated depending on task, so might need lot of small functions which do the right job. It is best case that user of a Struct/Class do not have direct access to all these functions.

```
struct Employee {
        string m_name;
        double m_salary;
    private:
        int m_age;

    public:
    void setAge(int age){
        if( !validAge(age) ) return;
        m_age = age;
    }

    private:
    bool validAge(int age){
        return age>0;
    }
};
```

Struct : Member Functions

Alternate (probably better) way to define member functions

```
struct Employee {
                string m_name;
                double m_salary;
        private:
                int m_age;

        public:
        void setAge(int age);

        private:
        bool validAge(int age);
};

void Employee::setAge(int age){
        m_age = age;
}

bool Employee::validAge(int age){
        return age>0;
}
```

Struct : Constructor

- Every object defined using Struct/Class needs a constructor to create the object.
- Generally used to initialize the data members.

```
struct Employee {
                string name;
                double salary;
        private:
                int age;
};

int main() {
        Employee emp1, emp2;
}
```

Is this invalid then?

Struct : Constructor

- Every object defined using Struct/Class needs a constructor to create the object.
- Generally used to initialize the data members.

Note that constructor doesn't have a return type (a special function!!)

<StructName>() → specification of default constructor

```
struct Employee {
		string name;
		double salary;
	private:
		int age;

	public:
	Employee(){}
};

int main() {
	Employee emp1, emp2;
}
```

Is this invalid then?

- It's valid. Compiler defines a default Constructor for us

Struct : Constructor

Can we define our own constructor?

-

If we write our own constructor, will the compiler still write a default one for us?

-

Can we define constructor with parameters?

-

Can we define multiple constructors?

-

Does ‘Employee’ struct have a default constructor?

-

```
struct Employee {
                string m_name;
                double m_salary;
        private:
                int m_age;

        public:
        Employee(string name){
                m_name = name;
        }

        Employee(int age){
                m_age = age;
        }
};
```

Struct : Constructor

Can we define our own constructor?

- *Yes!*

If we write our own constructor, will the compiler still write a default one for us?

- *No!*

Can we define constructor with parameters?

- *Yes!*

Can we define multiple constructors?

- *Yes!*

Does 'Employee' struct have a default constructor?

- *No. Default is one is with no parameters*
  => Employee() {...}

```
struct Employee {
                string m_name;
                double m_salary;
        private:
                int m_age;

        public:
        Employee(string name){
                m_name = name;
        }

        Employee(int age){
                m_age = age;
        }
};
```

Struct : Constructor

```
Employee emp;              // Will this compile?
                   -

Employee emp(“jack”);      // Will this compile?
                   -

Employee emp(100);         // Will this compile?
                   -

Employee emp(200.0);       // Will this compile?
                   -
```

```
struct Employee {
            string m_name;
            double m_salary;
      private:
            int m_age;

      public:
      Employee(string name){
            m_name = name;
      }

      Employee(int age){
            m_age = age;
      }
};
```

Struct : Constructor

```
Employee emp;              // Will this compile?
                      -    No, it doesn't have a default
                           constructor.

Employee emp("jack");      // Will this compile?
                      -    Yes.

Employee emp(100);         // Will this compile?
                      -    Yes.

Employee emp(200.0);       // Will this compile?
                      -    No, it doesn't have constructor with
                           double as a parameter.
```

```
struct Employee {
            string m_name;
            double m_salary;
      private:
            int m_age;

      public:
      Employee(string name){
            m_name = name;
      }

      Employee(int age){
            m_age = age;
      }
};
```

Struct : Constructor

```
Employee emp;            // Will this compile?
                   -
```

```
struct Employee {
        string m_name;
        double m_salary;
    private:
        int m_age;

    public:
    Employee(){
        m_salary = 0.0;
        m_age = 1;
    }

    Employee(string name){
        m_name = name;
    }

    Employee(int age){
        m_age = age;
    }
};
```

Struct : Constructor

```
Employee emp;            // Will this compile?
                      -  Yes.
```

```
struct Employee {
            string m_name;
            double m_salary;
      private:
            int m_age;

      public:
      Employee(){
            m_salary = 0.0;
            m_age = 1;
      }

      Employee(string name){
            m_name = name;
      }

      Employee(int age){
            m_age = age;
      }
};
```

Struct : Dynamic allocation

Dynamic objects are which created during run-time.
These are stored in the heap-memory.

Most of the times we might need to create objects during run-time
as we do not know full details during compile time.

```
struct Employee {
                string m_name;
                double m_salary;
        private:
                int m_age;

        public:
        Employee(){
                m_salary = 0.0;
                m_age = 1;
        }

        Employee(string name){
                m_name = name;
        }

        Employee(int age){
                m_age = age;
        }
};
```

Struct : Dynamic allocation

Dynamic objects are which created during run-time.
These are stored in the heap-memory.

Most of the times we might need to create objects during run-time as we do not know full details during compile time.

How can we create a dynamic struct/class object?

-

```
struct Employee {
            string m_name;
            double m_salary;
      private:
            int m_age;

      public:
      Employee(){
            m_salary = 0.0;
            m_age = 1;
      }

      Employee(string name){
            m_name = name;
      }

      Employee(int age){
            m_age = age;
      }
};
```

Struct : Dynamic allocation

Dynamic objects are which created during run-time.
These are stored in the heap-memory.

Most of the times we might need to create objects during run-time as we do not know full details during compile time.

How can we create a dynamic struct/class object?

- Using ‘new’ declaration

```
Employee * p = new Employee();
```

note that new always returns a pointer to the object created.

```
struct Employee {
            string m_name;
            double m_salary;
      private:
            int m_age;

      public:
      Employee(){
            m_salary = 0.0;
            m_age = 1;
      }

      Employee(string name){
            m_name = name;
      }

      Employee(int age){
            m_age = age;
      }
};
```

Struct : Dynamic allocation

Dynamic objects are which created during run-time.
These are stored in the heap-memory.

Most of the times we might need to create objects during run-time as we do not know full details during compile time.

How can we create a dynamic struct/class object?

- Using ‘new’ declaration

```
Employee * p = new Employee();
```

note that new always returns a pointer to the object created.

```
// Will this compile?
Employee * p = new Employee(“Jill”);
p->m_salary = 256.0;
```

```
struct Employee {
                string m_name;
                double m_salary;
        private:
                int m_age;

        public:
        Employee(){
                m_salary = 0.0;
                m_age = 1;
        }

        Employee(string name){
                m_name = name;
        }

        Employee(int age){
                m_age = age;
        }
};
```

Struct : Dynamic allocation

Dynamic objects are which created during run-time.
These are stored in the heap-memory.

Most of the times we might need to create objects during run-time as we do not know full details during compile time.

How can we create a dynamic struct/class object?

- Using ‘new’ declaration

```
Employee * p = new Employee();
```

note that new always returns a pointer to the object created.

```
// Will this compile? - Yes
Employee * p = new Employee("Jill");
p->m_salary = 256.0;
```

```
struct Employee {
		string m_name;
		double m_salary;
	private:
		int m_age;

	public:
	Employee(){
		m_salary = 0.0;
		m_age = 1;
	}

	Employee(string name){
		m_name = name;
	}

	Employee(int age){
		m_age = age;
	}
};
```

Struct : Dynamic allocation

```
int main() {
        Employee * p = new Employee(“Jill”);
        p->m_salary = 256.0;
        return 0;
}
```

Is something not right with our program?

```
struct Employee {
                string m_name;
                double m_salary;
        private:
                int m_age;

        public:
        Employee(){
                m_salary = 0.0;
                m_age = 1;
        }

        Employee(string name){
                m_name = name;
        }

        Employee(int age){
                m_age = age;
        }
};
```

Struct : Dynamic allocation

```
int main() {
        Employee * p = new Employee(“Jill”);
        p->m_salary = 256.0;
        return 0;
}
```

Is something not right with our program?

- Memory leaks, we haven’t deleted the memory allocated.

```
struct Employee {
                string m_name;
                double m_salary;
        private:
                int m_age;

        public:
        Employee(){
                m_salary = 0.0;
                m_age = 1;
        }

        Employee(string name){
                m_name = name;
        }

        Employee(int age){
                m_age = age;
        }
};
```

Struct : Dynamic allocation

```
int main() {
        Employee * p = new Employee(“Jill”);
        p->m_salary = 256.0;
        delete p;
        return 0;
}
```

Is something not right with our program?

- Memory leaks, we haven’t deleted the memory allocated.

Use ‘delete’ to free up the memory allocated.

- Call ‘delete’ only on dynamically allocated object’s pointer.

```
struct Employee {
                string m_name;
                double m_salary;
        private:
                int m_age;

        public:
        Employee(){
                m_salary = 0.0;
                m_age = 1;
        }

        Employee(string name){
                m_name = name;
        }

        Employee(int age){
                m_age = age;
        }
};
```

Struct : with Functions

You can pass struct/class as function parameters just like any other type.

```
void increaseSalary(Employee employee){
        employee.m_salary += 5.0;
}

int main() {
        Employee emp;
        emp.m_name = “William”;
        increaseSalary(emp);
        cout << “salary is ”<<emp.m_salary<<endl; // output ?
}
```

```
struct Employee {
                string m_name;
                double m_salary;
        private:
                int m_age;

        public:
        Employee(){
                m_salary = 0.0;
                m_age = 1;
        }
};
```

Struct : with Functions

You can pass struct/class as function parameters just like any other type.

```
void increaseSalary(Employee employee){
        employee.m_salary += 5.0;
}

int main() {
        Employee emp;
        emp.m_name = “William”;
        increaseSalary(emp);
        cout << “salary is ”<<emp.m_salary<<endl; // output ?
                                                     - 0.0, pass by value!
}
```

```
struct Employee {
                string m_name;
                double m_salary;
        private:
                int m_age;

        public:
        Employee(){
                m_salary = 0.0;
                m_age = 1;
        }
};
```

Struct : with Functions

```
void increaseSalary(Employee & employee){
        employee.m_salary += 5.0;
}

int main() {
        Employee emp;
        emp.m_name = “William”;
        increaseSalary(emp);
        cout << “salary is ”<<emp.m_salary<<endl; // output ?

}
```

```
struct Employee {
                string m_name;
                double m_salary;
        private:
                int m_age;

        public:
        Employee(){
                m_salary = 0.0;
                m_age = 1;
        }
};
```

Struct : with Functions

```
void increaseSalary(Employee & employee){
        employee.m_salary += 5.0;
}

int main() {
        Employee emp;
        emp.m_name = “William”;
        increaseSalary(emp);
        cout << “salary is ”<<emp.m_salary<<endl; // output ?
                                                    - 5.0, pass by reference!

}
```

```
struct Employee {
                string m_name;
                double m_salary;
        private:
                int m_age;

        public:
        Employee(){
                m_salary = 0.0;
                m_age = 1;
        }
};
```

Struct : with Functions

Passing struct/class pointers.

```
void increaseSalary(Employee * emp_p){
        emp_p->m_salary +=  5.0;
}

int main() {
        Employee emp;
        emp.m_name = “William”;
        increaseSalary(emp);
        cout << “salary is ”<<emp.m_salary<<endl; // output ?

}
```

```
struct Employee {
                string m_name;
                double m_salary;
        private:
                int m_age;

        public:
        Employee(){
                m_salary = 0.0;
                m_age = 1;
        }
};
```

Struct : with Functions

Passing struct/class pointers.

```
void increaseSalary(Employee * emp_p){
        emp_p->m_salary +=  5.0;
}

int main() {
        Employee emp;
        emp.m_name = “William”;
        increaseSalary(emp);
        cout << “salary is ”<<emp.m_salary<<endl; // output ?
                                - Compilation error!, expecting pointer
}
```

```
struct Employee {
                string m_name;
                double m_salary;
        private:
                int m_age;

        public:
        Employee(){
                m_salary = 0.0;
                m_age = 1;
        }
};
```

Struct : with Functions

Passing struct/class pointers.

```
void increaseSalary(Employee * emp_p){
        emp_p->m_salary +=  5.0;
}

int main() {
        Employee emp;
        emp.m_name = “William”;
        increaseSalary(&emp);
        cout << “salary is ”<<emp.m_salary<<endl; // output ?
}
```

```
struct Employee {
                string m_name;
                double m_salary;
        private:
                int m_age;

        public:
        Employee(){
                m_salary = 0.0;
                m_age = 1;
        }
};
```

Struct : with Functions

Passing struct/class pointers.

```
void increaseSalary(Employee * emp_p){
        emp_p->m_salary +=  5.0;
}

int main() {
        Employee emp;
        emp.m_name = “William”;
        increaseSalary(&emp);
        cout << “salary is ”<<emp.m_salary<<endl; // output ?
}
```

- 5.0, we are modifying using pointer.

```
struct Employee {
                string m_name;
                double m_salary;
        private:
                int m_age;

        public:
        Employee(){
                m_salary = 0.0;
                m_age = 1;
        }
};
```

Struct : with Functions

We can also pass struct/class as arguments to member functions.

```
struct Employee {
                string m_name;
        private:
                double m_salary;
                int m_age;

        public:
        Employee(string name){
                m_name = name;
                m_salary = 0.0;
                m_age = 1;
        }

        double getSalary(){
                return m_salary;
        }
};

struct UC {
        double m_budget;
        public:
        UC(){m_budget = 9000.0}

        bool isOnStrike(Employee * emp){
                double sal = emp->getSalary();
                return sal<100.0;
        }
}
```

Struct : with Functions

We can also pass struct/class as arguments to member functions.

```
int main() {
        UC ucla;
        Employee emp(“Einstein”);
        cout<< ucla.isOnStrike(&emp) << endl;
}
```

```
struct Employee {
                string m_name;
        private:
                double m_salary;
                int m_age;

        public:
        Employee(string name){
                m_name = name;
                m_salary = 0.0;
                m_age = 1;
        }

        double getSalary(){
                return m_salary;
        }
};

struct UC {
        double m_budget;
        public:
        UC(){m_budget = 9000.0}

        bool isOnStrike(Employee * emp){
                double sal = emp->getSalary();
                return sal<100.0;
        }
}
```

Struct : with Functions

We can also pass struct/class as arguments to member functions.

```
int main() {
        UC ucla;
        Employee emp("Einstein");
        cout<< ucla.isOnStrike(&emp) << endl;
}
```

What if we use const reference for Employee.
It means the function should not be able to change the Employee object pointed by.

Will this compile?

-

```
struct Employee {
                string m_name;
        private:
                double m_salary;
                int m_age;

        public:
        Employee(string name){
                m_name = name;
                m_salary = 0.0;
                m_age = 1;
        }

        double getSalary(){
                return m_salary;
        }
};

struct UC {
        double m_budget;
        public:
        UC(){m_budget = 9000.0;}

        bool isOnStrike(const Employee * emp){
                double sal = emp->getSalary();
                return sal<100.0;
        }
};
```

Struct : with Functions

We can also pass struct/class as arguments to member functions.

```
int main() {
        UC ucla;
        Employee emp(“Einstein”);
        cout<< ucla.isOnStrike(&emp) << endl;
}
```

What if we use const reference for Employee.
It means the function should not be able to change the Employee object pointed by.

Will this compile?

- No, The compiler doesn’t know if method getSalary() modifies Employee.

How can we fix this?

-

```
struct Employee {
                string m_name;
        private:
                double m_salary;
                int m_age;

        public:
        Employee(string name){
                m_name = name;
                m_salary = 0.0;
                m_age = 1;
        }

        double getSalary(){
                return m_salary;
        }
};

struct UC {
        double m_budget;
        public:
        UC(){m_budget = 9000.0;}

        bool isOnStrike(const Employee * emp){
                double sal = emp->getSalary();
                return sal<100.0;
        }
};
```

Struct : with Functions

We can also pass struct/class as arguments to member functions.

```
int main() {
        UC ucla;
        Employee emp("Einstein");
        cout<< ucla.isOnStrike(&emp) << endl;
}
```

What if we use const reference for Employee.
It means the function should not be able to change the Employee object pointed by.

Will this compile?

- No, The compiler doesn't know if method getSalary() modifies Employee.

How can we fix this?

- 'const' member functions

```
struct Employee {
                string m_name;
        private:
                double m_salary;
                int m_age;

        public:
        Employee(string name){
                m_name = name;
                m_salary = 0.0;
                m_age = 1;
        }

        double getSalary(){
                return m_salary;
        }
};

struct UC {
        double m_budget;
        public:
        UC(){m_budget = 9000.0;}

        bool isOnStrike(const Employee * emp){
                double sal = emp->getSalary();
                return sal<100.0;
        }
};
```

Struct : Const Member Functions

‘Const’ member functions are not supposed to modify the object.
Compiler knows that the object is not modified.

```
int main() {
        UC ucla;
        Employee emp(“Einstein”);
        cout<< ucla.isOnStrike(&emp) << endl;
}
```

What if we use const reference for Employee.
It means the function should not be able to change the Employee object pointed by.

Will this compile?

- No, The compiler doesn’t know if method getSalary() modifies Employee.

How can we fix this?

- ‘const’ member functions

```
struct Employee {
                string m_name;
        private:
                double m_salary;
                int m_age;

        public:
        Employee(string name){
                m_name = name;
                m_salary = 0.0;
                m_age = 1;
        }

        double getSalary() const {
                return m_salary;
        }
};

struct UC {
        double m_budget;
        public:
        UC(){m_budget = 9000.0;}

        bool isOnStrike(const Employee * emp){
                double sal = emp->getSalary();
                return sal<100.0;
        }
};
```

Struct : Const Member Functions

```
int main() {
        UC ucla;
        Employee emp("Einstein");
        cout<< ucla.isOnStrike(emp) << endl;
}
```

Will this compile?

-

```
struct Employee {
                string m_name;
        private:
                double m_salary;
                int m_age;

        public:
        Employee(string name){
                m_name = name;
                m_salary = 0.0;
                m_age = 1;
        }

        double getSalary() const {
                m_salary++;
                return m_salary;
        }
};

struct UC {
        double m_budget;
        public:
        UC(){m_budget = 9000.0}

        bool isOnStrike(const Employee * emp){
                double sal = emp->getSalary();
                return sal<100.0;
        }
}
```

Struct : Const Member Functions

```
int main() {
        UC ucla;
        Employee emp("Einstein");
        cout<< ucla.isOnStrike(emp) << endl;
}
```

Will this compile?

- No, const member function shouldn't modify any variable.

```
struct Employee {
                string m_name;
        private:
                double m_salary;
                int m_age;

        public:
        Employee(string name){
                m_name = name;
                m_salary = 0.0;
                m_age = 1;
        }

        double getSalary() const {
                m_salary++;
                return m_salary;
        }
};

struct UC {
        double m_budget;
        public:
        UC(){m_budget = 9000.0}

        bool isOnStrike(const Employee * emp){
                double sal = emp->getSalary();
                return sal<100.0;
        }
}
```

Struct : with Arrays

You can use struct/class with arrays, similar to any other type.

```
int main(){
        Employee emps[100]; // array of Employee's
        Employee * emp_ps[100]; // array of Employee Pointers
};
```

```
struct Employee {
                string m_name;
        private:
                double m_salary;
                int m_age;

        public:
        Employee(string name){
                m_name = name;
                m_salary = 0.0;
                m_age = 1;
        }

        double getSalary() const {
                return m_salary;
        }
};
```

Struct : with Arrays

You can also declare member variables which are arrays of type struct/class

```
int main(){
        UC ucla;
        ucla.addEmployee(“Alice”);
        ucla.addEmployee(“Bob”);
        return 0;
};
```

```
struct Employee {
                string m_name;
        private:
                double m_salary;
                int m_age;

        public:
        Employee(string name){
                m_name = name;
                m_salary = 0.0;
                m_age = 1;
        }

        double getSalary() const {
                return m_salary;
        }
};

Struct UC {
        Employee * emp_ps[100];
        int n=0;

        addEmployee(string name){
                Employee * p = new Employee(name);
                emp_ps[n] = p;
                n++;
        }

};
```

Struct : with Arrays

You can also declare member variables which are arrays of type struct/class

```
int main(){
          UC ucla;
          ucla.addEmployee("Alice");
          ucla.addEmployee("Bob");
          return 0;
};
```

Is something not right with our program?

-

```
struct Employee {
                    string m_name;
          private:
                    double m_salary;
                    int m_age;

          public:
          Employee(string name){
                    m_name = name;
                    m_salary = 0.0;
                    m_age = 1;
          }

          double getSalary() const {
                    return m_salary;
          }
};

Struct UC {
          Employee * emp_ps[100];
          int n=0;

          addEmployee(string name){
                    Employee * p = new Employee(name);
                    emp_ps[n] = p;
                    n++;
          }

};
```

Struct : with Arrays

You can also declare member variables which are arrays of type struct/class

```
int main(){
          UC ucla;
          ucla.addEmployee("Alice");
          ucla.addEmployee("Bob");
          return 0;
};
```

Is something not right with our program?

- Memory leaks, we haven't deleted the memory allocated.

How can we fix this?

-

```
struct Employee {
                    string m_name;
          private:
                    double m_salary;
                    int m_age;

          public:
          Employee(string name){
                    m_name = name;
                    m_salary = 0.0;
                    m_age = 1;
          }

          double getSalary() const {
                    return m_salary;
          }
};

Struct UC {
          Employee * emp_ps[100];
          int n=0;

          addEmployee(string name){
                    Employee * p = new Employee(name);
                    emp_ps[n] = p;
                    n++;
          }

};
```

Struct : with Arrays

You can also declare member variables which are arrays of type struct/class

```
int main(){
            UC ucla;
            ucla.addEmployee(“Alice”);
            ucla.addEmployee(“Bob”);
            return 0;
};
```

Is something not right with our program?

- Memory leaks, we haven’t deleted the memory allocated.

How can we fix this?

- Destructors

```
struct Employee {
                        string m_name;
            private:
                        double m_salary;
                        int m_age;

            public:
            Employee(string name){
                        m_name = name;
                        m_salary = 0.0;
                        m_age = 1;
            }

            double getSalary() const {
                        return m_salary;
            }
};

Struct UC {
            Employee * emp_ps[100];
            int n=0;

            addEmployee(string name){
                        Employee * p = new Employee(name);
                        emp_ps[n] = p;
                        n++;
            }

};
```

Struct : Destructor

Destructors are special functions which help cleanup memory in classes/structs. Destructor of a class is called when the objects goes out of scope or when ‘delete’ is called on its pointer.

If you do not specify a destructor, the compiler will write a default one for you.

~<StructName>() → specification of destructor

```
struct Employee {
                string m_name;
        private:
                double m_salary;
                int m_age;

        public:
        Employee(string name){
                m_name = name;
                m_salary = 0.0;
                m_age = 1;
        }

        ~Employee(){}
};
```

Struct : Destructor

The fix to our problem

```
int main(){
        UC ucla;
        ucla.addEmployee(“Alice”);
        ucla.addEmployee(“Bob”);
        return 0;
};
```

```
struct Employee {
                string m_name;
        private:
                double m_salary;
                int m_age;

        public:
        Employee(string name){
                m_name = name;
                m_salary = 0.0;
                m_age = 1;
        }

};

Struct UC {
        Employee * emp_ps[100];
        int n=0;

        addEmployee(string name){
                Employee * p = new Employee(name);
                emp_ps[n] = p;
                n++;
        }

        ~UC() {
            while(n-1>0){
                delete emp_ps[n-1];
                n- -;
            }
        }
};
```

Struct : Destructor

The fix to our problem

```
int main(){
            UC ucla;
            ucla.addEmployee(“Alice”);
            ucla.addEmployee(“Bob”);
            return 0;
};
```

Can we define destructor with parameters?

-

```
struct Employee {
                        string m_name;
            private:
                        double m_salary;
                        int m_age;

            public:
            Employee(string name){
                        m_name = name;
                        m_salary = 0.0;
                        m_age = 1;
            }

            ~Employee(){} // default destructor.
};

Struct UC {
            Employee * emp_ps[100];
            int n=0;

            addEmployee(string name){
                        Employee * p = new Employee(name);
                        emp_ps[n] = p;
                        n++;
            }

            ~UC() {
                  while(n-1>0){
                        delete emp_ps[n-1];
                        n- -;
                  }
            }
}
```

Struct : Destructor

The fix to our problem

```
int main(){
        UC ucla;
        ucla.addEmployee(“Alice”);
        ucla.addEmployee(“Bob”);
        return 0;
};
```

Can we define destructor with parameters?

- *No!, we don’t need them*

Can we define multiple destructors?

-

```
struct Employee {
                string m_name;
        private:
                double m_salary;
                int m_age;

        public:
        Employee(string name){
                m_name = name;
                m_salary = 0.0;
                m_age = 1;
        }

        ~Employee(){} // default destructor.
};

Struct UC {
        Employee * emp_ps[100];
        int n=0;

        addEmployee(string name){
                Employee * p = new Employee(name);
                emp_ps[n] = p;
                n++;
        }

        ~UC() {
            while(n-1>0){
                delete emp_ps[n-1];
                n- -;
            }
        }
}
```

Struct : Destructor

The fix to our problem

```
int main(){
        UC ucla;
        ucla.addEmployee(“Alice”);
        ucla.addEmployee(“Bob”);
        return 0;
};
```

Can we define destructor with parameters?

- *No!, we don’t need them*

Can we define multiple destructors?

- *No! Every class/struct must have one and only one.*

```
struct Employee {
                string m_name;
        private:
                double m_salary;
                int m_age;

        public:
        Employee(string name){
                m_name = name;
                m_salary = 0.0;
                m_age = 1;
        }

        ~Employee(){} // default destructor.
};

Struct UC {
        Employee * emp_ps[100];
        int n=0;

        addEmployee(string name){
                Employee * p = new Employee(name);
                emp_ps[n] = p;
                n++;
        }

        ~UC() {
            while(n-1>0){
                delete emp_ps[n-1];
                n- -;
            }
        }
}
```

Struct : Destructor

The fix to our problem

```
int main(){
        UC ucla;
        ucla.addEmployee(“Alice”);
        ucla.addEmployee(“Bob”);
        return 0;
};
```

Can we define destructor with parameters?
- *No!, we don’t need them*

Can we define multiple destructors?
- *No! Every class/struct must have one and only one.*

Destructor of a class is called when the object goes out of scope or when ‘delete’ is called on its pointer.

```
struct Employee {
                string m_name;
        private:
                double m_salary;
                int m_age;

        public:
        Employee(string name){
                m_name = name;
                m_salary = 0.0;
                m_age = 1;
        }

        ~Employee(){} // default destructor.
};

Struct UC {
        Employee * emp_ps[100];
        int n=0;

        addEmployee(string name){
                Employee * p = new Employee(name);
                emp_ps[n] = p;
                n++;
        }

        ~UC() {
            while(n-1>0){
                delete emp_ps[n-1];
                n- -;
            }
        }
}
```

Summary :

Pointers

- Declaration, Initialization
- *, & operators
- Pointers with arrays
- Pointers with functions
- Pointer axioms/arithmetic.

Struct/Classes

- Defining, Declaration, Initialization, '.' dot operation
- Pointer to struct, '->' operator
- Public/Private, Class vs Struct
- Member functions : inside struct, outside struct, const member functions
- Constructors : Default, with arguments
- Dynamic allocation, 'delete' operator
- Destructor
- Structs with Functions
- Array of Structs, Struct Pointers

References -
https://cplusplus.com/doc/tutorial/pointers/

Summary :

Pointers

- Declaration, Initialization
- *, & operators
- Pointers with arrays
- Pointers with functions
- Pointer axioms/arithmetic.

Struct/Classes

- Defining, Declaration, Initialization, '.' dot operation
- Pointer to struct, '->' operator
- Public/Private, Class vs Struct
- Member functions : inside struct, outside struct, const member functions
- Constructors : Default, with arguments
- Dynamic allocation, 'delete' operator
- Destructor
- Structs with Functions
- Array of Structs, Struct Pointers

Questions??

References -
https://cplusplus.com/doc/tutorial/pointers/