

REFERENCES, POINTERS AND STRUCTS

Problem Solving with Computers-I

<https://ucsb-cs16-wi17.github.io/>

C++

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

int main(){
    cout<<"Hola Facebook\n";
    return 0;
}
```



What is the output of this function?

A. 10 20

B. 20 10

```
void swapValue(int x, int y){  
    int tmp = x;  
    x = y;  
    y = tmp;  
}
```

```
int main() {  
    int a=10, b=20;  
    swapValue( a, b);  
    cout<<a<<" "<<b<<endl;  
}
```

Modify the function to swap the values of a and b: use pointers

```
void swapValue(int x, int y){  
    int tmp = x;  
    x = y;  
    y = tmp;  
}  
  
int main() {  
    int a=10, b=20;  
    swapValue( a, b);  
    cout<<a<<" "<<b<<endl;  
}
```

Draw the pointer diagram for your code

Segmentation faults (aka segfault)

- Segfault- your program has crashed!
- What caused the crash?
 - Read or write to a memory location that either doesn't exist or you don't have permission to access
- Avoid segfaults in your code by
 - Always initializing a pointer to null upon declaration
 - Performing a null check before dereferencing it
 - Avoid redundant null checks by specifying pre and post conditions for functions that use pointers

Code 1:

```
char *p;  
int y;  
p = &y;
```

Code 2:

```
int *p;  
*p = 5;
```

Q: Which of the following is true about the above code?

	Code 1	Code 2
A	Compile time warning/error	Compile time error
B	Runtime error	Compile time error
C	Compile time warning/error	Runtime error
D	Runtime error	Run time error
E	None of the above	

References in C++

```
int main() {  
    int d = 5;  
    int &e = d;  
}
```


A reference in C++ is an alias for another variable

A. d 

e 

C. d 
e 

B. d 

e 

D. This code causes an error

References in C++

```
int main() {  
    int d = 5;  
    int & e = d;  
    int f = 10;  
    e = f;  
}
```

How does the diagram change with this code?

A. 



C. 

B. 



D. Other or error

Pointers and references: Draw the diagram for this code

```
int a = 5;  
int & b = a;  
int* pt1 = &a;
```

What are three ways
to change the value
of 'a' to 42?

Call by reference: Modify to correctly swap a and b

```
void swapValue(int x, int y){  
    int tmp = x;  
    x = y;  
    y = tmp;  
}
```

```
int main() {  
    int a=10, b=20;  
    swapValue( a, b);  
    cout<<a<<" "<<b<<endl;  
}
```

C++ structures

- A **struct** is a data structure composed of simpler data types.

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

```
void PrintPoint(struct Point pt)  
{  
    cout<< pt.x << " " << pt.y);  
}
```

```
int main()  
{  
    struct Point p;  
    //Initialize p  
  
}
```

Pointers to structures

The C arrow operator (\rightarrow) dereferences and extracts a structure field with a single operator.

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

```
void PrintPoint(struct Point *pt){
```

```
}
```

```
int main() {  
    struct Point p;  
    //Initialize p  
    p.x = 10;  
    p.y = 20;
```

```
}
```

References to structures

```
void setPoint(struct Point &pt) {

}

void PrintPoint(struct Point &pt) {

}

int main(){
    struct Point p;
                                //Initialize p

}
```

Program layout in memory at runtime

A generic model for memory



Low address

Text

Global and static
data

Dynamic memory

High address

Two important facts about Pointers

- 1) A pointer can only point to one type –(basic or derived) such as `int`, `char`, a `struct`, another pointer, etc
- 2) After declaring a pointer: `int *ptr;`
`ptr` doesn't actually point to anything yet. We can either:
 - make it point to something that already exists, or
 - allocate room in memory for something new that it will point to
 - Null check before dereferencing

Complex declarations in C/C++

How do we decipher declarations of this sort?

```
int **arr[];
```

Read

- * as “pointer to” (always on the left of identifier)
- [] as “array of” (always to the right of identifier)
- () as “function returning” (always to the right ...)

For more info see:

http://ieng9.ucsd.edu/~cs30x/rt_lt.rule.html

Complex declarations in C/C++

Right-Left Rule

```
int **arr [];
```

Step 1: Find the identifier

Step 2: Look at the symbols to the right of the identifier. Continue right until you run out of symbols *OR* hit a *right* parenthesis ")"

Step 3: Look at the symbol to the left of the identifier. If it is not one of the symbols '*', '(', '[' just say it. Otherwise, translate it into English using the table in the previous slide. Keep going left until you run out of symbols *OR* hit a *left* parenthesis "(".

Repeat steps 2 and 3 until you've formed your declaration.

Illegal combinations include:

[]() - cannot have an array of functions

()() - cannot have a function that returns a function

()[] - cannot have a function that returns an array

Complex declarations in C/C++

```
int i;  
int *i;  
int a[10];  
int f( );  
int **p;  
int (*p)[ ];  
int (*fp)( );  
int *p[ ];  
int af[ ]( );  
int *f( );  
int fa()[ ];  
int ff()( );  
int (**ppa)[ ];  
int (*apa[ ])[ ] ;
```

Next time

- Arrays of structs
- Dynamic memory allocation