CS 31 Discussion 1J

ABDULLAH-AL-ZUBAER IMRAN

WEEK 8: MEMORY MANAGEMENT AND POINTERS

Recap

○ Class

○ Class constructors

○ Const modifier

○ Class declaration and definition separated

Discussion Objectives

Review and practice things covered during lectures

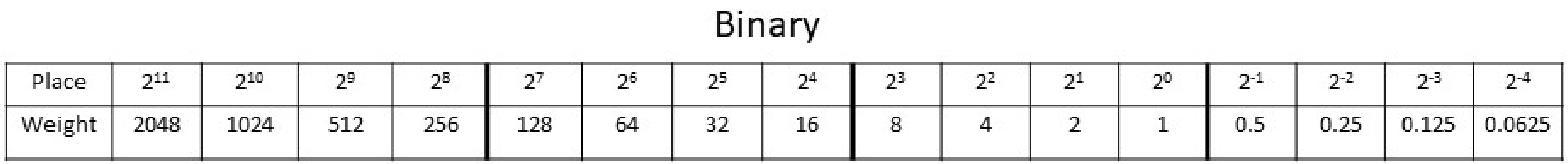
- Number Systems
- Memory management
- Pointers
  - Pointer and Arrays
  - Pointer Arithmetic
  - Pointer to Pointer

- Coding examples

- Worksheet 7
- Project 6

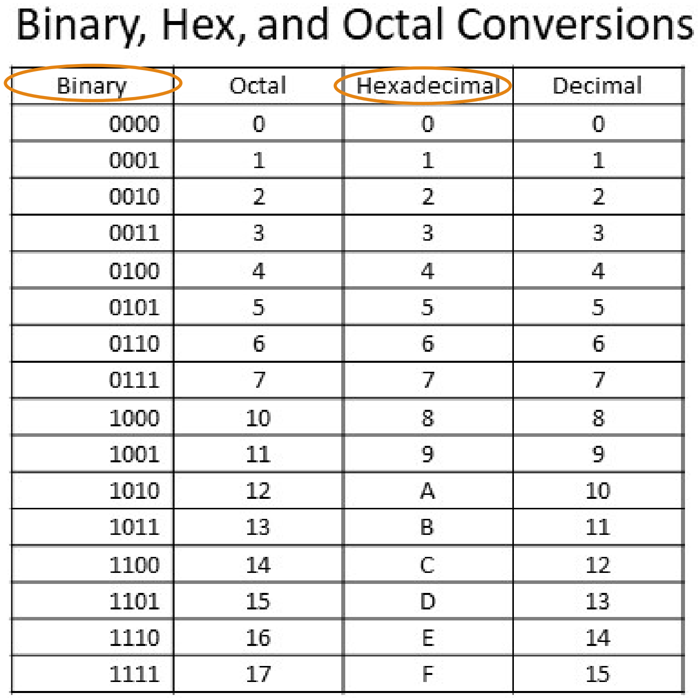
Time for you to ask questions!

Number Systems



Binary

Place	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}
Weight	2048	1024	512	256	128	64	32	16	8	4	2	1	0.5	0.25	0.125	0.0625



Binary, Hex, and Octal Conversions

Binary	Octal	Hexadecimal	Decimal
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	A	10
1011	13	B	11
1100	14	C	12
1101	15	D	13
1110	16	E	14
1111	17	F	15

Methodology

1. Convert from decimal to binary

   Divide the decimal by the largest binary weight it is divisible by and place a “1” in that column. Then select the next largest weight, if it is divisible put a “1” in that column otherwise place a “0” in the column. Continue until all the columns have either a “1” or “0” resulting in a binary expression.

2. Convert from decimal to hex.

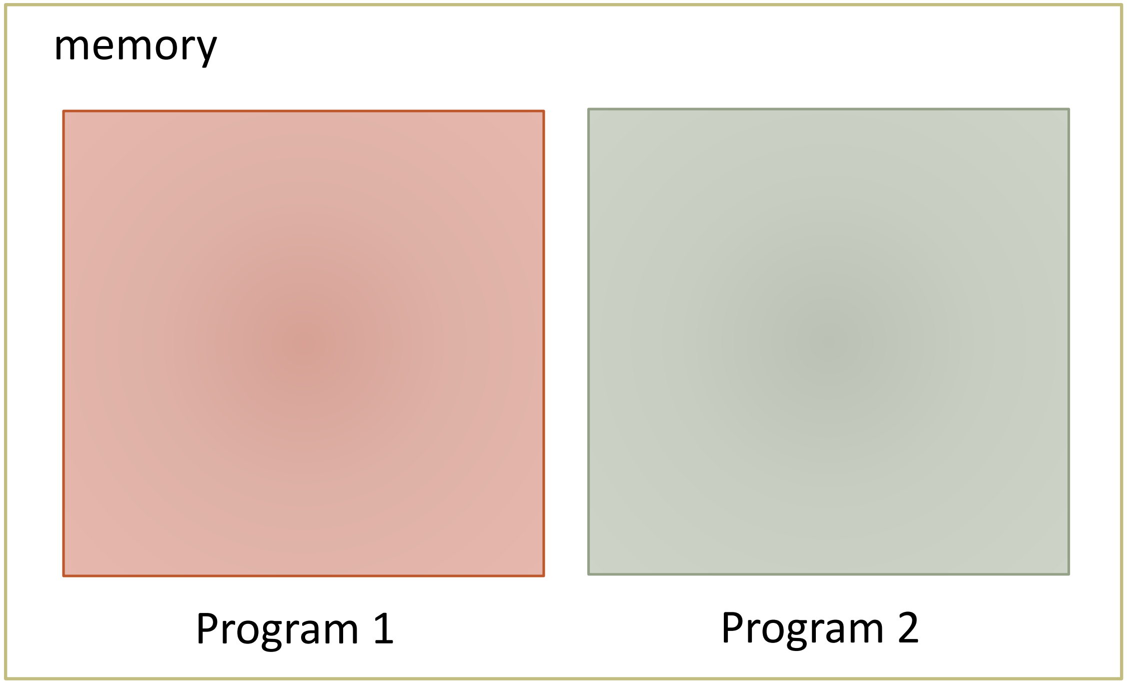
   Convert to binary first, then group the binary in groups of 4 beginning on the right working to the left. For each group determine the hex value based on the table to the left.

3. Convert to octal

   Convert to binary first, then group the binary in groups of 3 beginning on the right working to the left. For each group determine the octal value based on the table to the left.

Memory Management

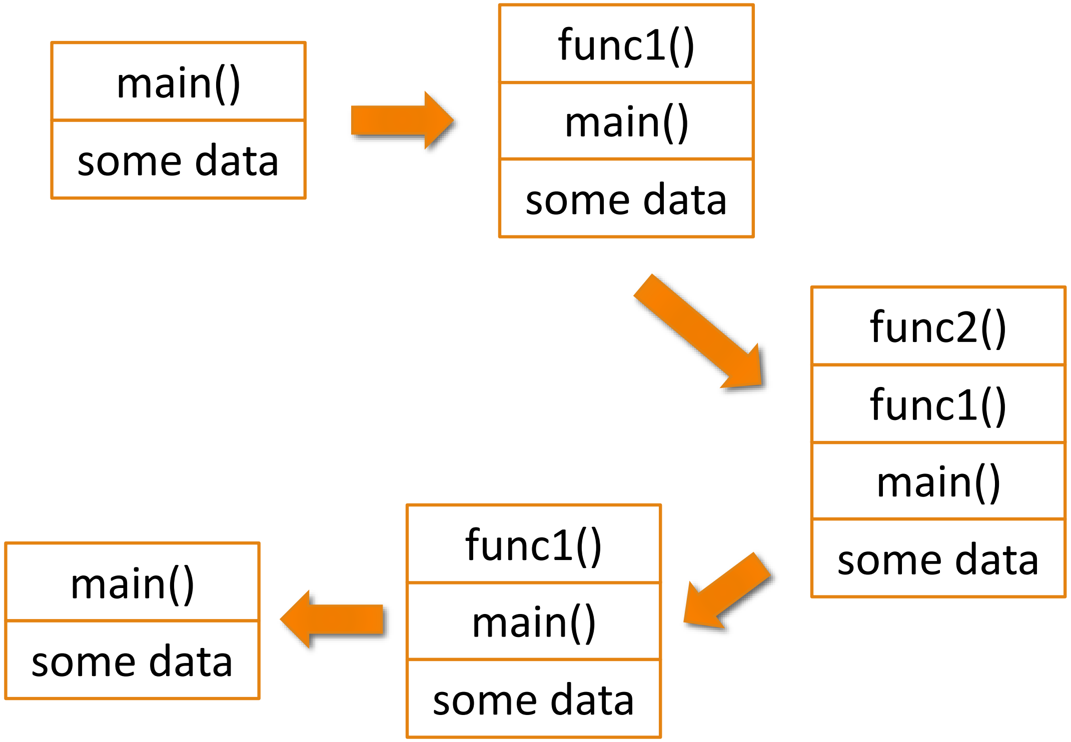
When the program gets executed, it gets some amount of memory allocated for use.



Memory Management

Consider this program

```
int main() {
  func1(); // call func1()
}
void func1() {
  ...
  func2(); // call func2()
}
```



main()
some data

func1()
main()
some data

func2()
func1()
main()
some data

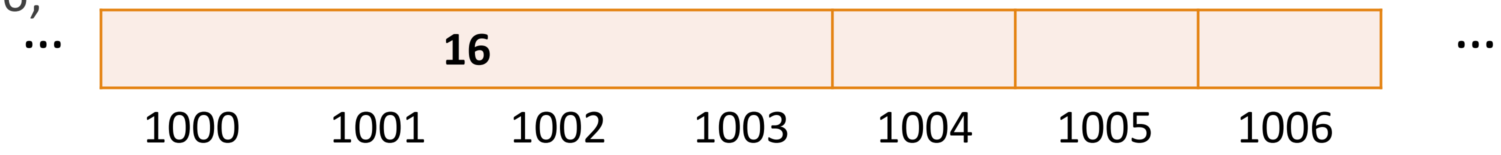
func1()
main()
some data

main()
some data

Memory Management

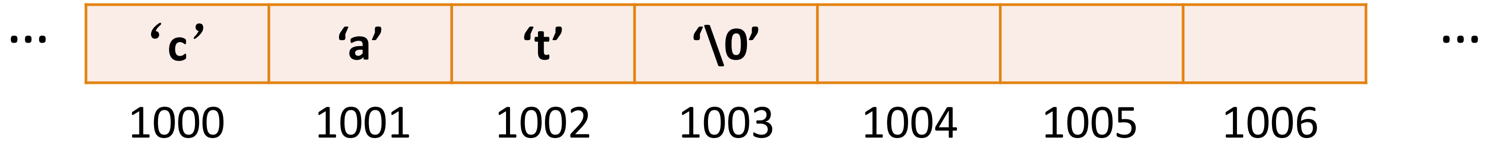
Every variable you create during the program execution gets its own space in some location within the memory. And every location is marked with a unique **address**.

int x = 16;



…	**16**							…
	1000	1001	1002	1003	1004	1005	1006	

char c[] = “cat”

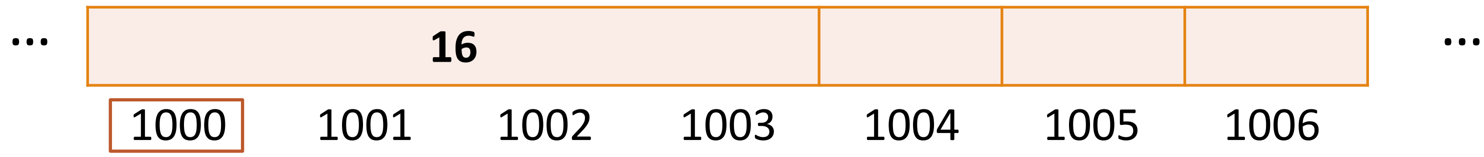


…	**‘c’**	**‘a’**	**‘t’**	**‘\0’**				…
	1000	1001	1002	1003	1004	1005	1006	

Pointers

The address-of operator (&): get the memory address of the expression to the right of the ampersand.

int x = 16;



… 16 …

1000 1001 1002 1003 1004 1005 1006

cout << &x << endl;

Pointers

Pointers store memory addresses and are assigned a type corresponding to the type of the variable that they point to.

<type>* <name> // declares a pointer of the given <type> and calls it <name>.

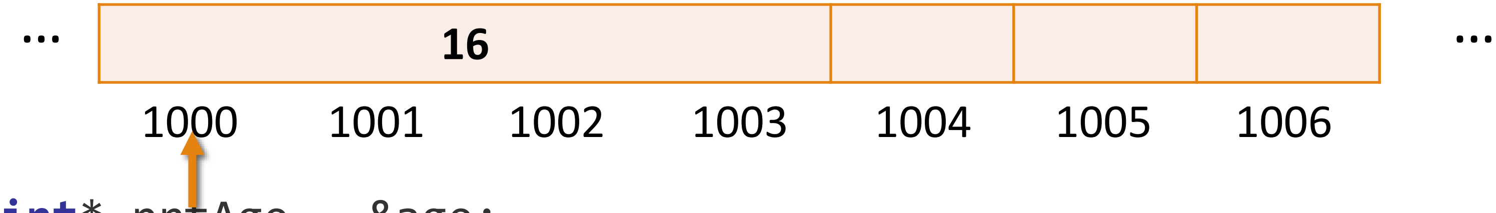
```
int* ptrAge;
bool* ptrValid;
char* ptrName;
```

To **initialize** pointers

```
int age = 40;
int* prtAge = &age;
```

or

```
int* ptrAge;
ptrAge = &age;
```



… 16 …

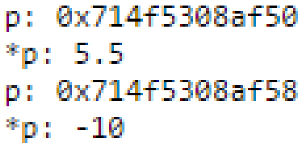
1000 1001 1002 1003 1004 1005 1006

```
int* prtAge = &age;
```

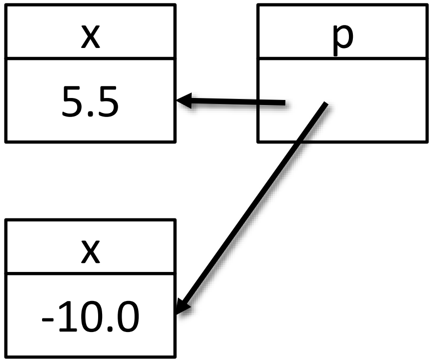
Pointers

The dereference operator (*): to dereference a pointer to get the variable pointed by the pointer.

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

int main() {
  double x, y;
// normal double variables
  double *p;
// a pointer to a double variable
  x = 5.5;
  y = -10.0;
  p = &x;
// assign x’s memory address to p (make p point to x)
  cout << "p: " << p << endl;
  cout << "*p: " << *p << endl;
  p = &y;
  cout << "p: " << p << endl;
  cout << "*p: " << *p << endl;
  return 0;
}
```

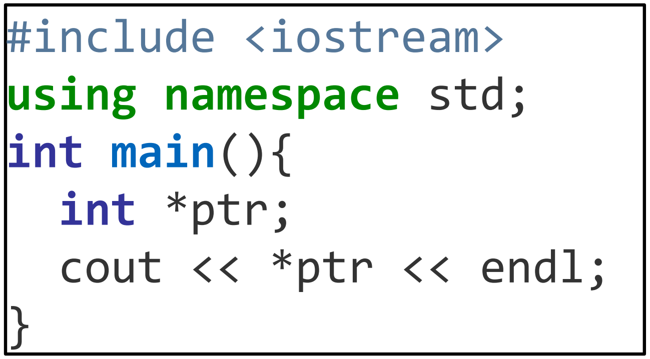


```
p: 0x714f5308af50
*p: 5.5
p: 0x714f5308af58
*p: -10
```

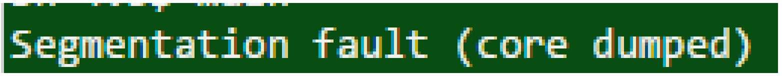


Pointers

Question: Will the code compile? If so, what's the output?



```cpp
#include <iostream>
using namespace std;
int main(){
  int *ptr;
  cout << *ptr << endl;
}
```



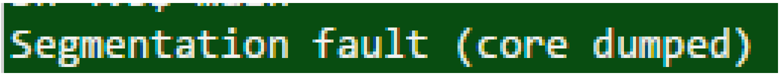
```
Segmentation fault (core dumped)
```

Be careful! Uninitialized pointers can lead to undefined behavior or illegal memory accesses when they haven't been assigned somewhere first.
A special keyword `nullptr` that represents "the pointer that points at nothing".

Pointers

Question: Will the code compile? If so, what's the output?

```
#include <iostream>
using namespace std;
int main(){
  int *ptr = nullptr;
  cout << *ptr << endl;
}
```

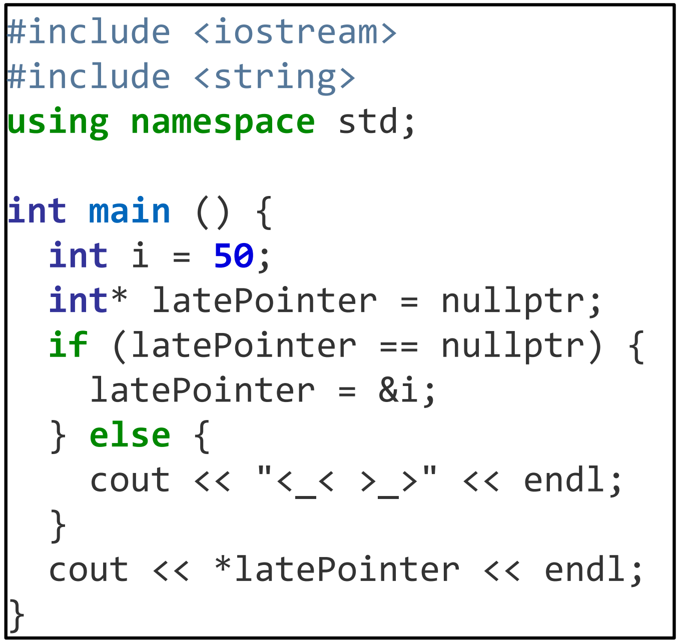


```
Segmentation fault (core dumped)
```

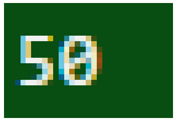
Attempting to dereference a `nullptr` will result in undefined behavior.

Pointers

We can check to make sure a pointer is or is not null pointer.



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

int main () {
  int i = 50;
  int* latePointer = nullptr;
  if (latePointer == nullptr) {
    latePointer = &i;
  } else {
    cout << "<_< >_>" << endl;
  }
  cout << *latePointer << endl;
}
```



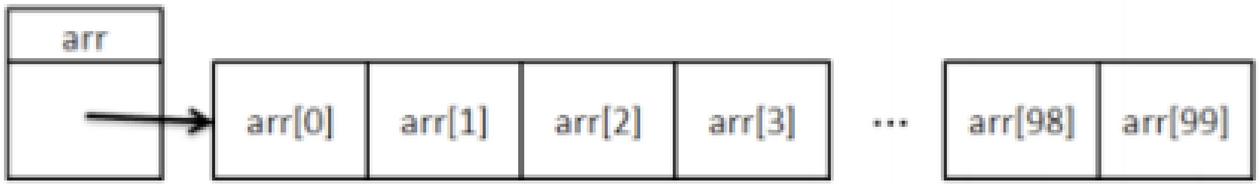
```
50
```

Pointers and Arrays

int arr[100];

arr is actually a pointer(int*)

Special for arr, the pointee can’t change.

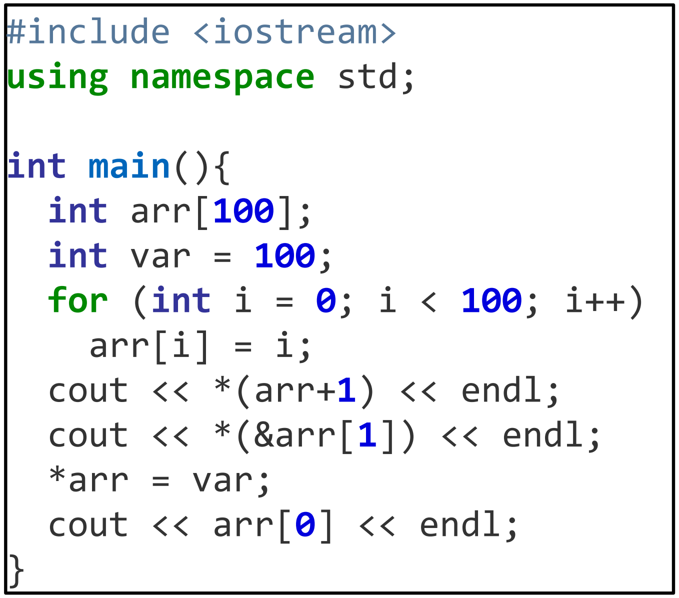


In order to get the value of arr[1]

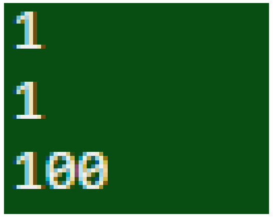
- arr[1]
- *(arr+1)

Pointers and Arrays

Question: Will the code compile? If so, what's the output?



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

int main(){
  int arr[100];
  int var = 100;
  for (int i = 0; i < 100; i++)
    arr[i] = i;
  cout << *(arr+1) << endl;
  cout << *(&arr[1]) << endl;
  *arr = var;
  cout << arr[0] << endl;
}
```

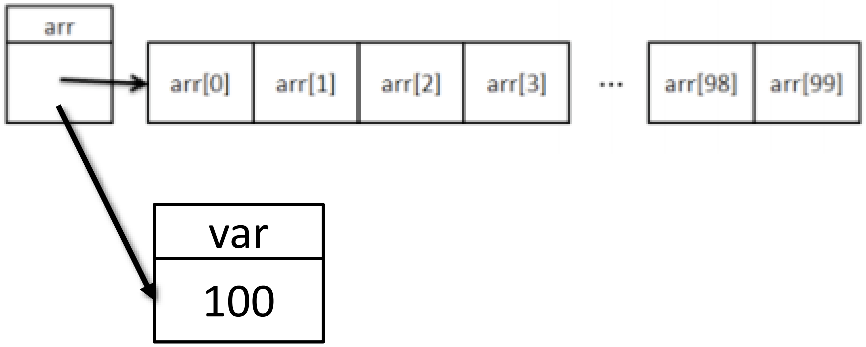


```
1
1
100
```

Pointers and Arrays

Question: Will the code compile? If so, what's the output?

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

int main(){
  int arr[100];
  int var = 100;
  for (int i = 0; i < 100; i++)
    arr[i] = i;
  cout << *(arr+1) << endl;
  cout << *(&arr[1]) << endl;
  arr = &var;
  cout << arr[0] << endl;
}
```

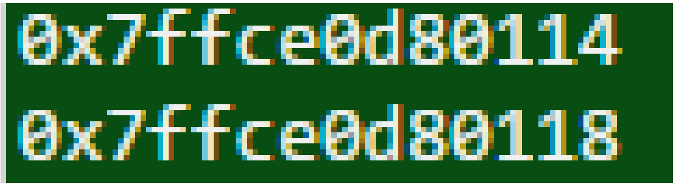


What about arr[1]?
arr+1 = ???

Array elements are located contiguously in memory.

Pointer Arithmetic

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

int main(){
  int arr[100];
  int var = 100;
  for (int i = 0; i < 100; i++)
    arr[i] = i;
  cout << arr+1 << endl;
  cout << arr+2 << endl;
}
```



```
0x7ffce0d80114
0x7ffce0d80118
```

arr is pointing to the “integer”.
A integer is of 4 bytes on this platform.

Pointer Arithmetic

Subtraction and addition to pointers is well defined, such that if I say `(ptr + i),` it means "*refer to the address i times x bytes away from ptr,*" where x is the size of the type of ptr.

Pointers are NOT defined on multiplication or division!

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

int main () {
  double d[] = {1.1, 2.2, 3.3, 4.4, 5.5};
  double* ptr = d; cout << *(ptr * 2) << endl;
}
```

✗

Pointers and Arrays

You can treat an array variable like a pointer – well, it is a pointer. Therefore, the following are equivalent:

```
int findFirst(const string a[], int n, string target);
int findFirst(const string* a, int n, string target);
```

Recall

- Pass by value

  ```
  int foo(int n);
  ```

- Pass by reference

  ```
  int foo(int &n);
  ```

- Pass by pointer

  ```
  int foo(int a[]);          int foo(int* a);
  ```

Program Challenge

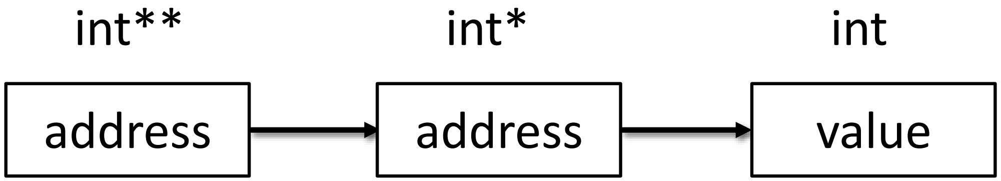
The following is one possible implementation of `findFirst()`. Can you modify it such that it doesn't use brackets?

```
int findFirst(const string a[], int n, string target) {
  for (int i = 0; i < n; i++)
    if (a[i] == target)
      return i;
  return -1;
}

int findFirst(const string* a, int n, string target) {
}
```

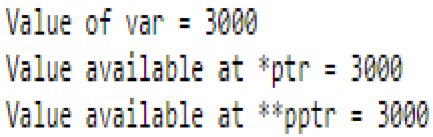
```
string a[5] = {"home", "marge", "bart", "marge", "lisa"};
cout << findFirst(a, 5, "marge") << endl;
cout << findFirst(a + 2, 3, "marge") << endl;
```

Pointer to pointer

int** var;



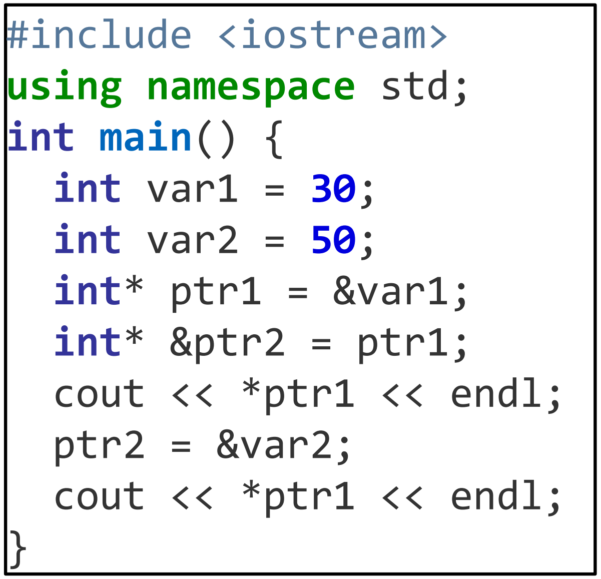
```
#include <iostream>
using namespace std;
int main() {
  int var;
  int *ptr;
  int **pptr;
  var = 3000;
  ptr = &var;
  pptr = &ptr;
  cout << "Value of var = " << var << endl;
  cout << "Value available at *ptr = " << *ptr << endl;
  cout << "Value available at **pptr = " << **pptr << endl;
}
```



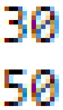
```
Value of var = 3000
Value available at *ptr = 3000
Value available at **pptr = 3000
```

Reference to Pointer

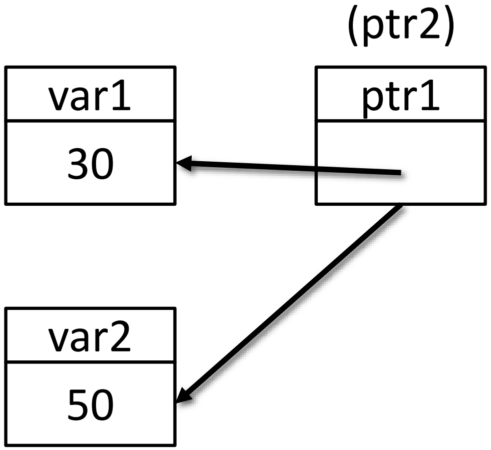
int* &ptr;



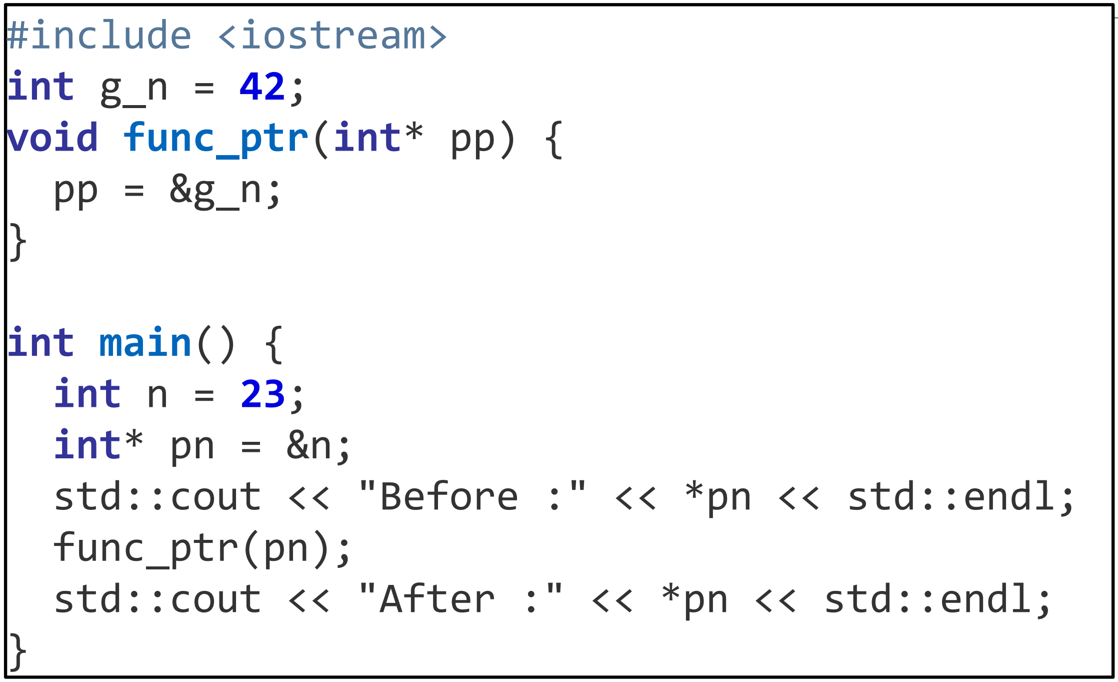
```
#include <iostream>
using namespace std;
int main() {
  int var1 = 30;
  int var2 = 50;
  int* ptr1 = &var1;
  int* &ptr2 = ptr1;
  cout << *ptr1 << endl;
  ptr2 = &var2;
  cout << *ptr1 << endl;
}
```



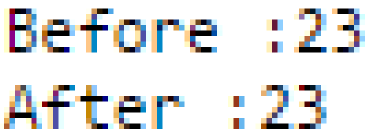
```
30
50
```



Why do we need them?



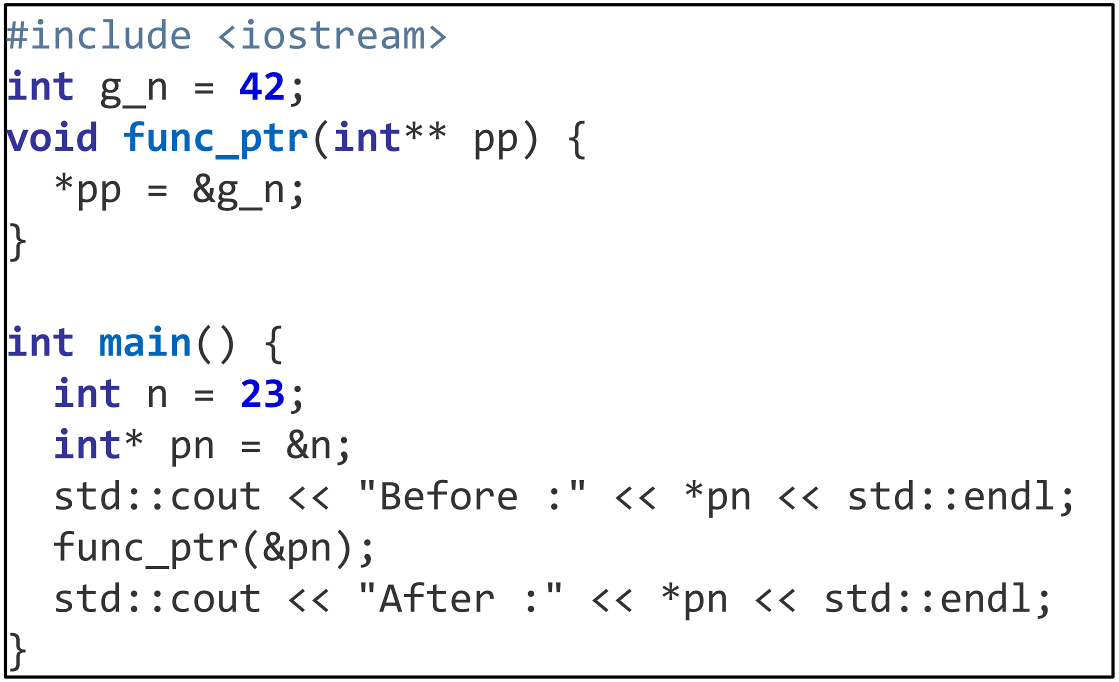
```cpp
#include <iostream>
int g_n = 42;
void func_ptr(int* pp) {
  pp = &g_n;
}

int main() {
  int n = 23;
  int* pn = &n;
  std::cout << "Before :" << *pn << std::endl;
  func_ptr(pn);
  std::cout << "After :" << *pn << std::endl;
}
```



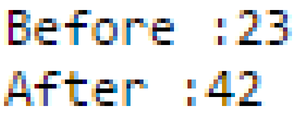
```
Before :23
After :23
```

Question: how can I get 42 after calling the function?

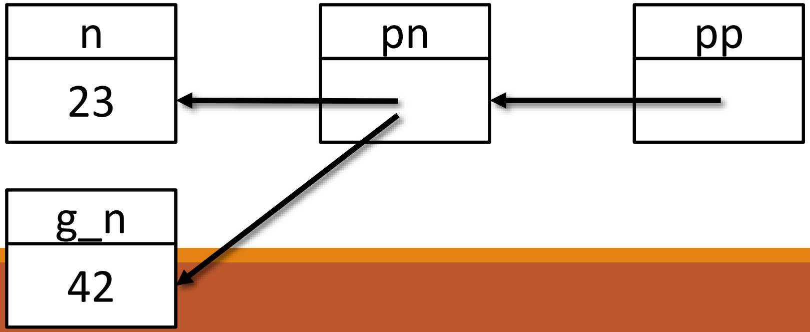
Solution1: Pointer to Pointer



```
#include <iostream>
int g_n = 42;
void func_ptr(int** pp) {
  *pp = &g_n;
}

int main() {
  int n = 23;
  int* pn = &n;
  std::cout << "Before :" << *pn << std::endl;
  func_ptr(&pn);
  std::cout << "After :" << *pn << std::endl;
}
```

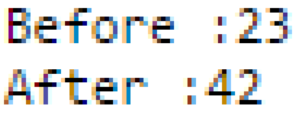


```
Before :23
After :42
```

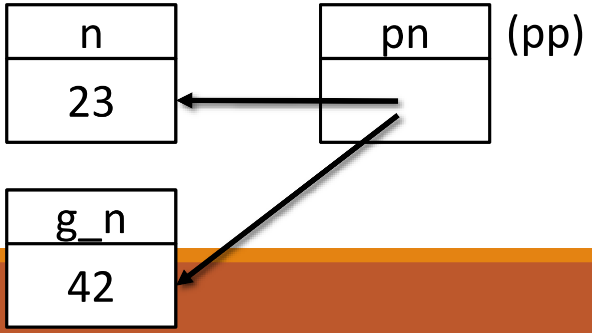


Solution2: Reference to Pointer

```
#include <iostream>
int g_n = 42;
void func_ptr(int* &pp) {
  pp = &g_n;
}

int main() {
  int n = 23;
  int* pn = &n;
  std::cout << "Before :" << *pn << std::endl;
  func_ptr(pn);
  std::cout << "After :" << *pn << std::endl;
}
```



```
Before :23
After :42
```



Project6

- More like a homework
- Designed to help you master pointers (six problems)
- **Time due: 9:00 PM Wednesday, March 4th**

Problem 1c: The smallest function is correct, but the main function has a problem. Explain why it may not work, and show a way to fix it. Your fix must be to the main function only; you must not change the smallest function in any way.

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

void smallest(int value1, int value2, int * resultPtr)
{
   if( value1 < value2 )
   {
         *resultPtr = value1;
   }
   else
   {
         *resultPtr = value2;
   }
}

int main()
{
   int* p;
   smallest(15, 20, p);
   cout << "The smallest value is " << *p << endl;
   return( 0 );
}
```

Problem 1e. This program is supposed to write 1 1 2 3 5 8 13 21 but it probably does not. What is the program doing that is incorrect? (We're not asking you explain why the incorrect action leads to the particular outcome it does, and we're not asking you to propose a fix to the problem.)

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

int fibonacci( int n )
{
    int tmp;
    int a = 1;
    int b = 1;

    for (int i = 0; i < n-2; i ++)
    {
         tmp = a+b;
        a = b;
         b = tmp;
    }
    return b;
}
```

```
int* computeFibonacciSequence(int& n)
{
    int arr[8];
    n = 8;
    for (int k = 0; k < n; k++)
    {
        arr[k] = fibonacci( k+1 );
    }
    return arr;
}


int main()
{
    int m;
    int* ptr = computeFibonacciSequence(m);
    for (int i = 0; i < m; i++)
    {
        cout << ptr[i] << ' ';
    }
    return( 0 );
}
```

Problem 6: Write a function named deleteDigits that accepts one character pointer as a parameter and returns no value.

The parameter must be a C-string. This function must remove all of the digit character letters from the string. The resulting string must be a valid C-string. Your function must declare no more than one local variable in addition to the parameter; that additional variable must be of a pointer type. Your function must not use any square brackets and must not use the strlen or strcpy library functions.

```
int main()
{
   char msg[100] = "Happy 2019!";
   deleteDigits(msg);
   cout << msg << endl;       // prints:   Happy !
}
```

Thanks!

Questions?

Today's discussion slides can be found at

https://github.com/zubaerimran/W20-CS31-1J/blob/master/week8/winter20_cs31_w8.pdf

Some of the materials presented have been taken from earlier TA discussions