CS 31 Discussion

ABDULLAH-AL-ZUBAER IMRAN

WEEK 7: MEMORY MANAGEMENT & POINTERS

Discussion Objectives

Review and practice things covered during lectures

- Number Systems
- Multi-dimensional Arrays
- Memory Management
- Pointers
- Project5

Programming Challenge

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.

Array

Multi-dimensional arrays

```
int i[ROWS][COLUMNS];

int i[ROWS][COLUMNS] = {
      {row_00, row_01, ...},
      {row_10, row11, ...},
      ...,
      {row_i0, row_i1, ...}};

some2DArray[i][j];
```

```
#include <iostream>

using namespace std;
int main () {
  int i[][3] = {
    {1, 2, 3},
    {4, 5, 6},
    {20, -1, 0} };
}
```

Arrays in a Function

```
void fibo10(int fib[]);
```

Note that the size of fib is not specified, you can explicitly pass the size in the function.

```
void fibo10(int fib[], int n);
```

Now after you learnt about 2D arrays.

```
void fibo10(int fib[][], int n);
```

C Strings

An array of C strings

```
char s[10][20];
```

In s, we can store up to ____ C strings, and each C string can be at most ____ characters long.

10

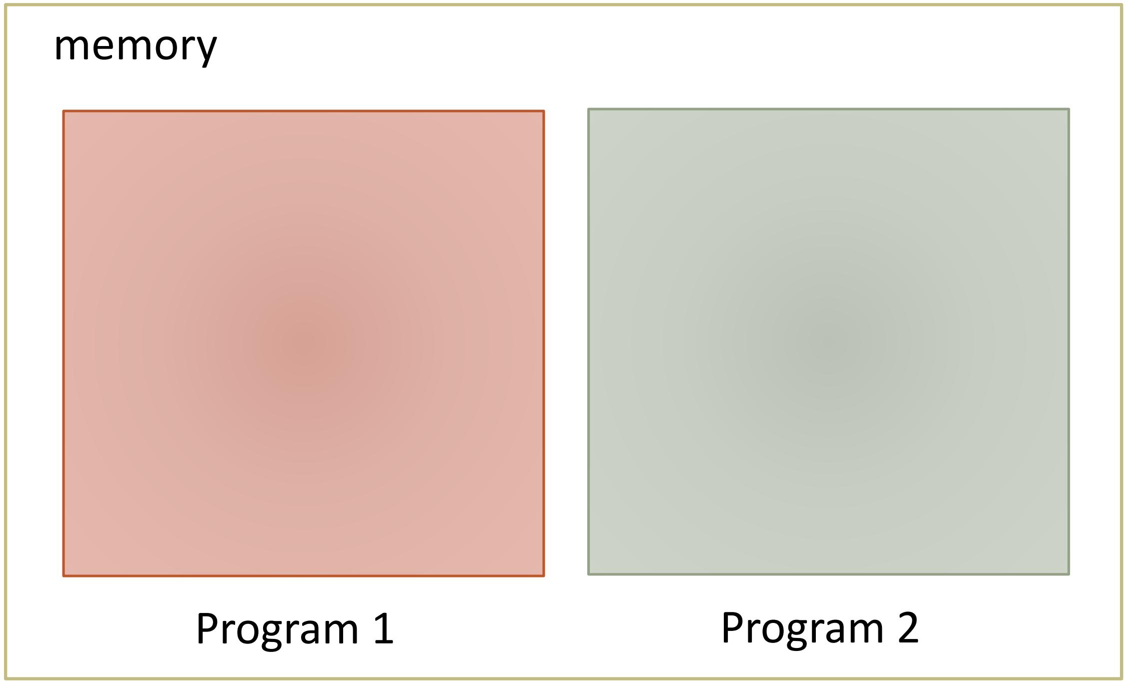
19

```
s[3];
// refer to the string in position 3

s[3][5];
// refer to the letter in position 5 of the string in position 3
```

Memory Management

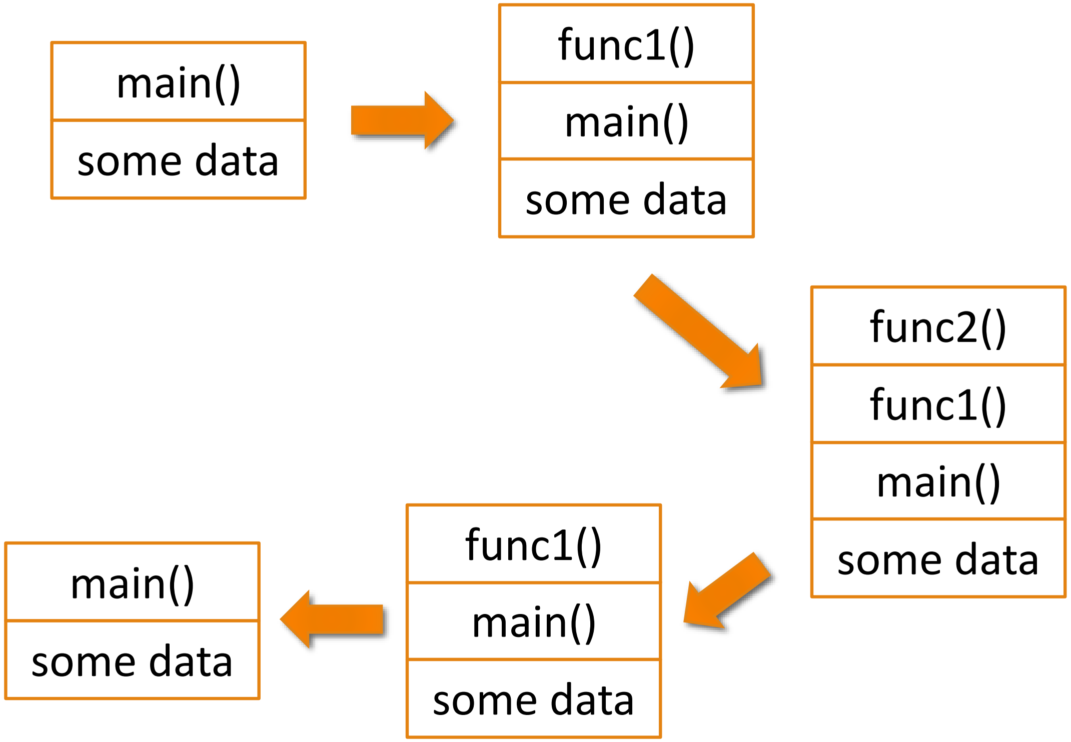
When the program gets executed, it gets some amount of memory allocated for use.



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

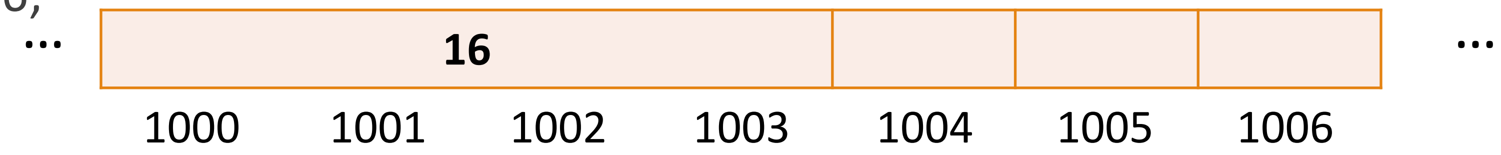
func1()
main()
some data

main()
some data

Memory Management

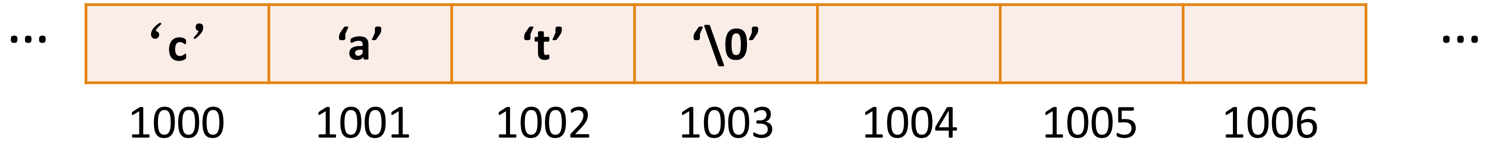
Every variable you create during the program execution gets its own space in some location within the memory. And every location is marked with a unique **address**.

int x = 16;



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

char c[] = “cat”

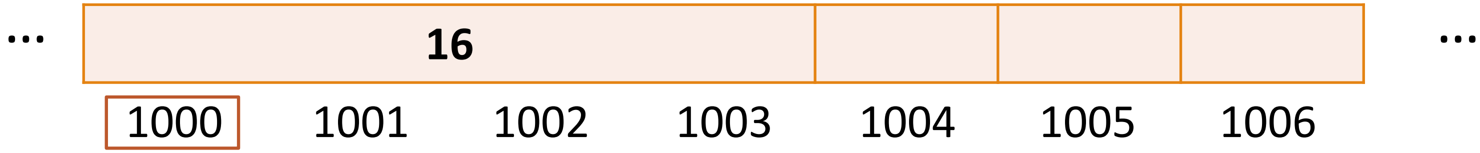


…	**‘c’**	**‘a’**	**‘t’**	**‘\0’**				…
	1000	1001	1002	1003	1004	1005	1006	

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>.

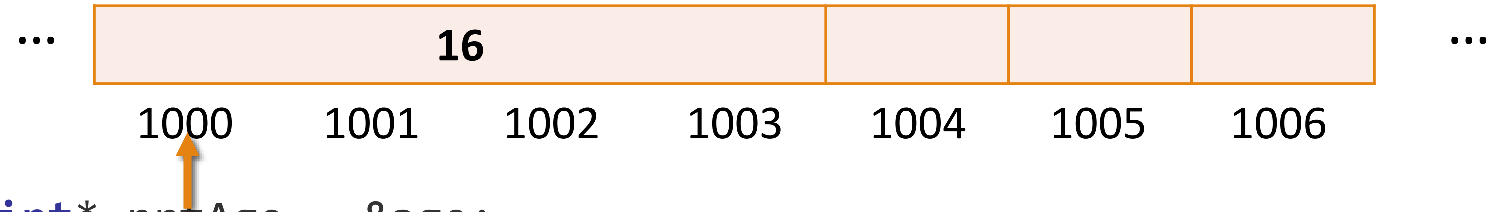
```
int* ptrAge;
bool* ptrValid;
char* ptrName;
```

To **initialize** pointers

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

or

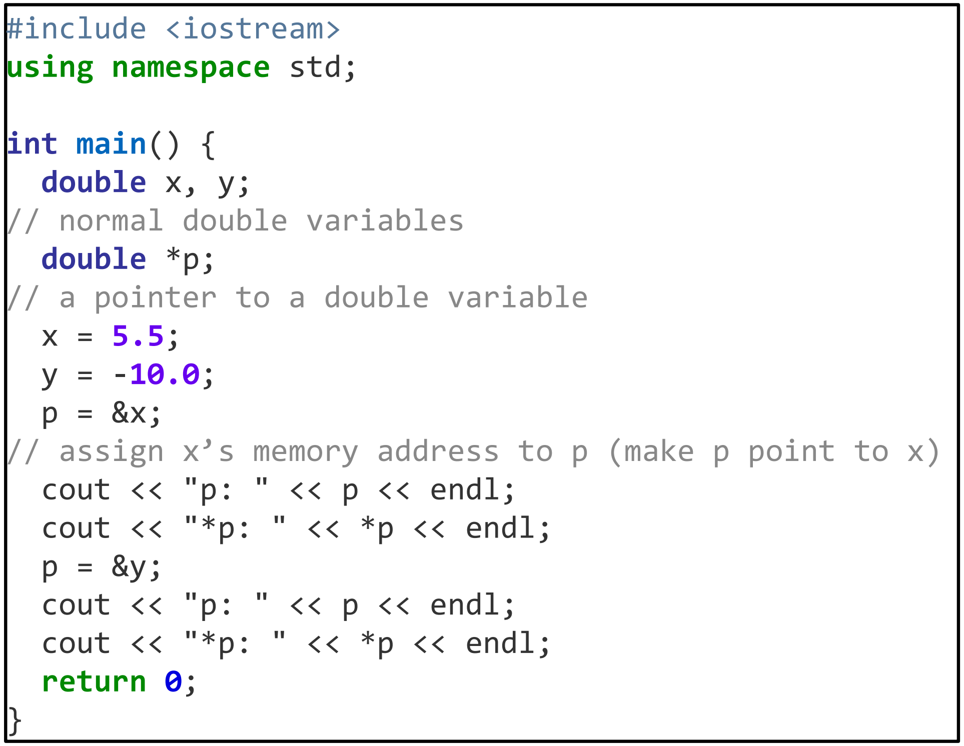
```
int* ptrAge;
ptrAge = &age;
```



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

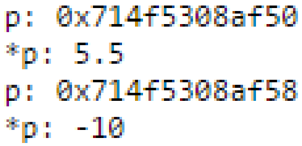
Pointers

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

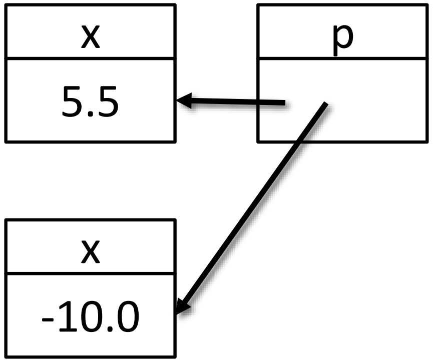


```
#include <iostream>
using namespace std;

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;
}
```



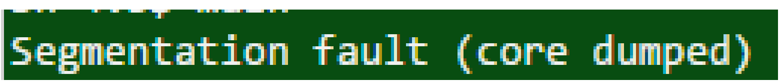
```
p: 0x714f5308af50
*p: 5.5
p: 0x714f5308af58
*p: -10
```



Pointers

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

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

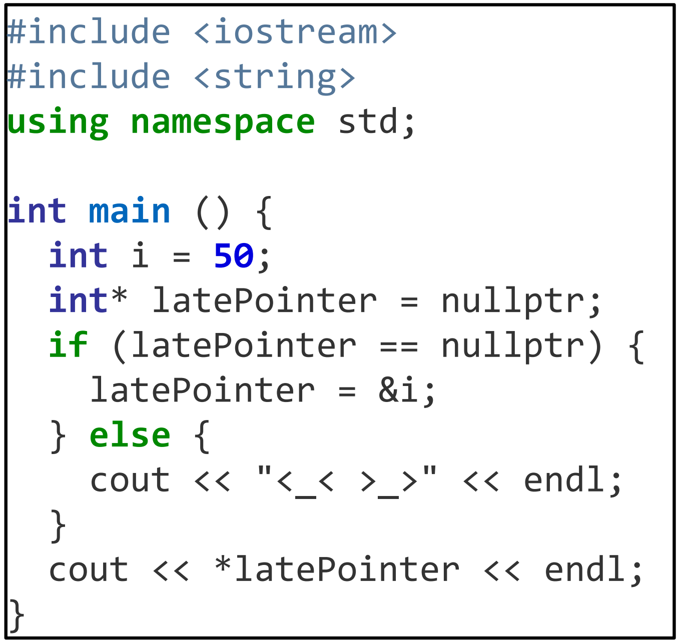


```
Segmentation fault (core dumped)
```

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".

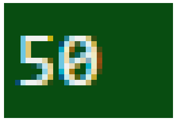
Pointers

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;
}
```



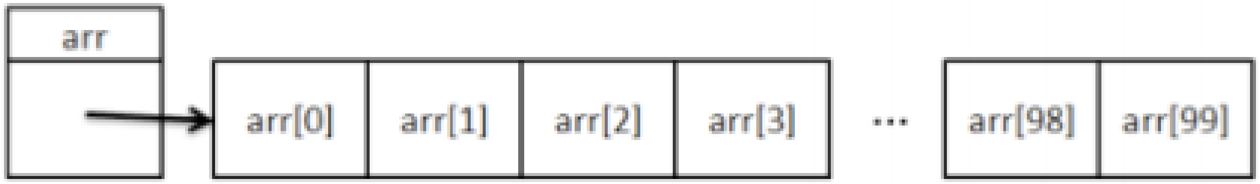
```
50
```

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]

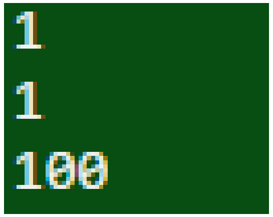
- arr[1]
- *(arr+1)

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;
}
```



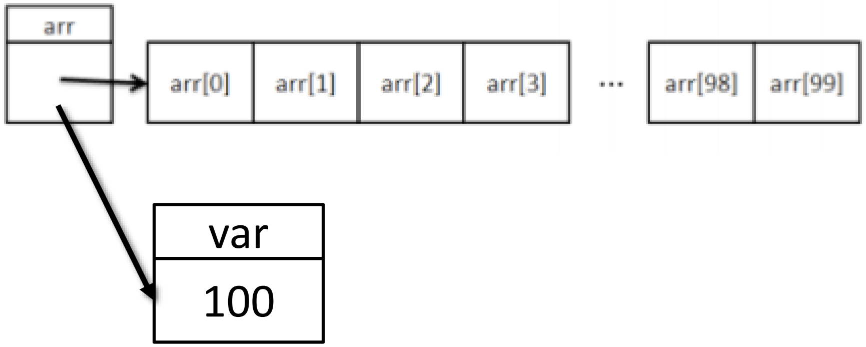
```
1
1
100
```

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;
}
```



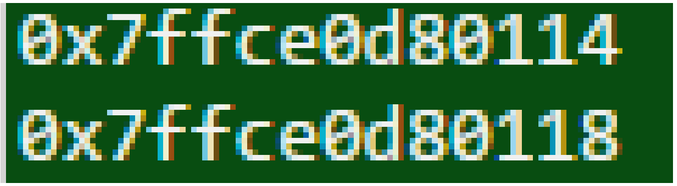
What about arr[1]?
arr+1 = ???

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

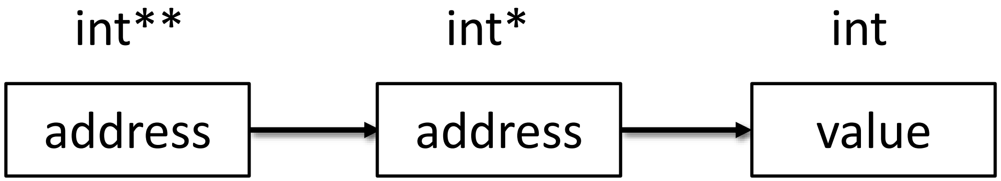
```
int foo(int &n);
```

- Pass by pointer

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

Pointer to pointer

int** var;



int**

address

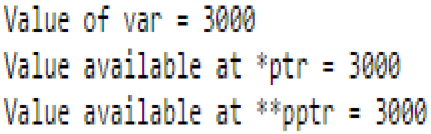
int*

address

int

value

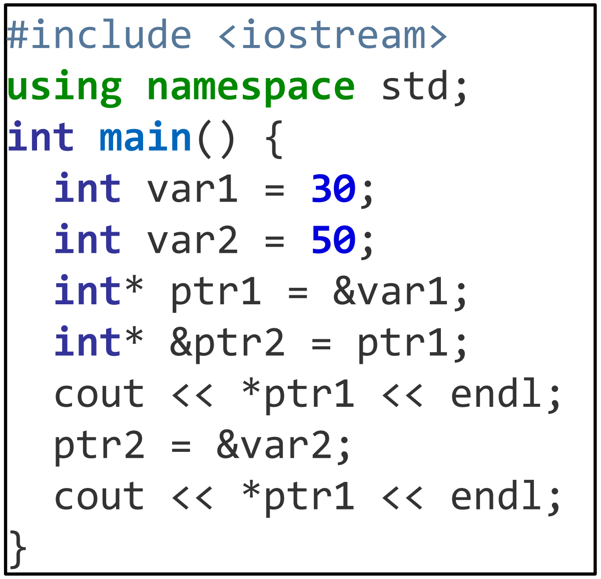
```
#include <iostream>
using namespace std;
int main() {
  int var;
  int *ptr;
  int **pptr;
  var = 3000;
  ptr = &var;
  pptr = &ptr;
  cout << "Value of var = " << var << endl;
  cout << "Value available at *ptr = " << *ptr << endl;
  cout << "Value available at **pptr = " << **pptr << endl;
}
```



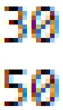
```
Value of var = 3000
Value available at *ptr = 3000
Value available at **pptr = 3000
```

Reference to Pointer

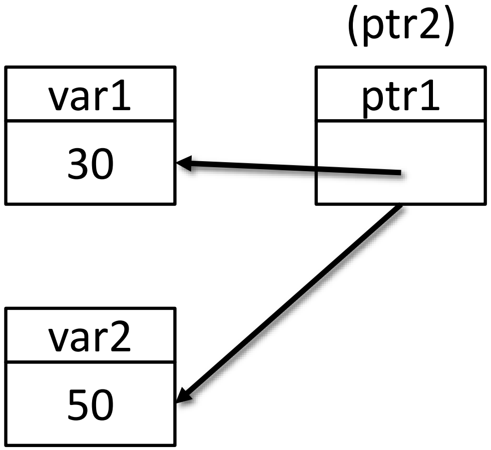
int* &ptr;



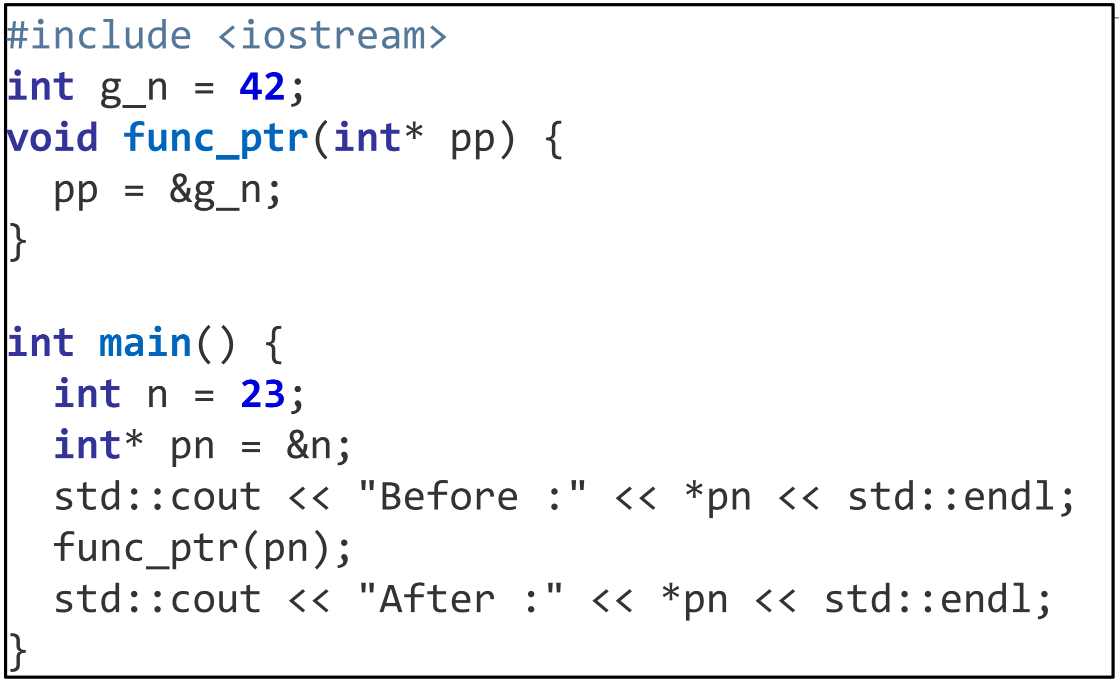
```
#include <iostream>
using namespace std;
int main() {
  int var1 = 30;
  int var2 = 50;
  int* ptr1 = &var1;
  int* &ptr2 = ptr1;
  cout << *ptr1 << endl;
  ptr2 = &var2;
  cout << *ptr1 << endl;
}
```



```
30
50
```

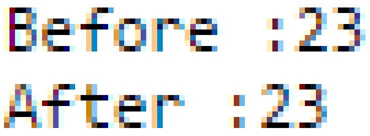


Why do we need them?



```
#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;
}
```



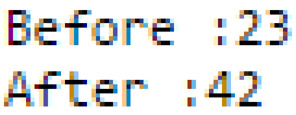
```
Before :23
After :23
```

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

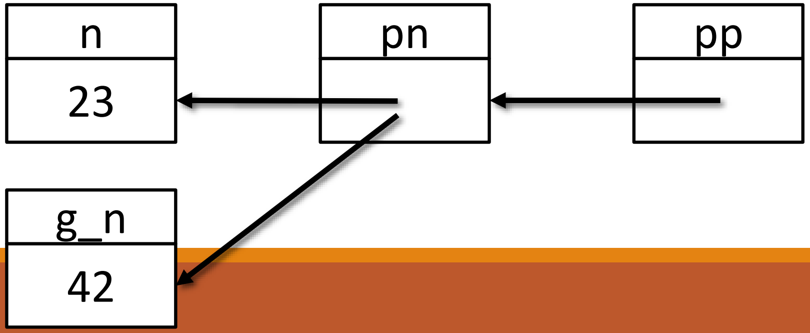
Solution1: Pointer to Pointer

```
#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;
}
```



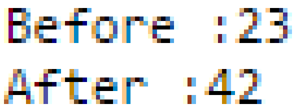
```
Before :23
After :42
```



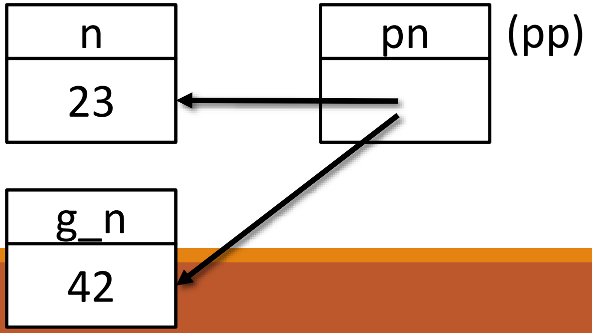
Solution2: Reference to Pointer

```
#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
```



Project 5

int countMatches(const string a[], int n, string target)→Return the number of strings in the array that are equal to target.

int detectMatch(const string a[], int n, string target)→Return the position of a string in the array that is equal to target.

bool detectSequence(const string a[], int n, string target, int& begin, int& end)→Find the earliest occurrence in a of one or more consecutive strings that are equal to target.

int detectMin(const string a[], int n)→Return the position of a string in the array such that that string is <= every string in the array.

int moveToBack(string a[], int n, int pos)→Eliminate the item at position pos by copying all elements after it one place to the left.

int moveToFront(string a[], int n, int pos)→Eliminate the item at position pos by copying all elements before it one place to the right.

Project5

- The program you turn in must use C strings
- Put the following line in your program before any of your #includes:

  #define _CRT_SECURE_NO_WARNINGS

- Function: bool decrypt(const char ciphertext[], const char crib[]);
- Incremental Development: start from a simple portion, write new functions for repetitive tasks, etc.
- You turn in: decrypt.cpp, report.docs (notable obstacles, pseudocode, and test cases with actual data)
- Time due: 11 PM Wednesday, May 22

```
runtest("Hirdd ejsy zu drvtry od.\nO'z fodvtrry.\n", "my secret");
runtest("Hirdd ejsy zu drvtry od.\nO'z fodvtrry.\n", "shadow");
```

```
====== my secret
hiESS ejsT MY SECRET oS.
o'M foSCREET.
Return value: true
====== shadow
Return value: false
```

Thanks!

Questions?

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