REFERENCES ARRAYS AND POINTERS

OVERVIEW

Fundamental type (examples):

- Void
- Arithmetic types: int, double, etc.
- char

Compound types (a type that is defined in terms of another type) in C++:

- **Functions**
- **Structs** and classes
- **References** – not independent type, an alias
- Arrays
- Pointers

REFERENCES

REFERENCE DECLARATION

```
int i = 10;
int &j = i, k = i; // j is reference to int, k is int

//both int &j = i; and int& j = i; are valid
```

A reference declaration declares an alias to an already existing object

There are no references to void

There are no references or pointers to references

There are no arrays of references

CALL-BY-REFERENCE PARAMETERS

When a function is called, its arguments are substituted for the formal parameters

There are 2 mechanisms for substituting the parameters:

Call-by-value – The formal parameters are initialized with the **value** of the variable passed as the arguments to the function

Call-by-reference – The corresponding argument in the function call **must be a variable**, the variable will be substituted for the formal parameter and any changes made to the formal parameter inside the function body will also change the value of the variable that was passed as the argument.

- To make a formal parameter (in the function definition), append an & (ampersand) to its type name.

- When the function is called it is not given the values of the variables, it is given the **memory location**

ARRAYS

ARRAYS

- Arrays consist of contiguously allocated elements of a type

  Example:

  `int arr[100];` //100 integers beginning with index 0

- If there are N contiguous elements (of the same type)
  - Elements are numbered 0,1…..(N-1) (indexes)
  - Elements can be accessed with [ ], example: arr[6] or arr[N-2]
  - Without an initializer, every element in the array is uninitialized

ARRAYS

Behaves like a list of variables with a uniform naming mechanism and can be declared in a single line of code.

Static array –size is determined when declared

Partially filled array – static array and keep track of the number of filled elements in the array

dynamic array – size is determined while the program is running

STATIC ARRAYS

- size is determined when declared

```
int score[5]; //declares an array of 5 integers

//the five variables are score[0], score[1], score[2], score[3], score[4]


//partially filled array example (STATIC ARRAY)
int numbers[50];

int numberCount = 0; //numberCount will keep track of how many elements
are in the array
```

ARRAYS

Behaves like a list of variables with a uniform naming mechanism and can be declared in a single line of code.

index – the number in the square brackets.
The indexed variables are the elements of the array

```
int idList[20]; //20 integers
double testScores[20];//20 doubles
char letterGrades[20];//20 characters
string names[20]; //20 strings
```

ARRAYS

Array example:

```
//reads in five scores and shows how much each score differs from the highest score
#include <iostream>
using namespace std;

int main()
{
      int score[5], i, max;

      cout << "Enter a score: ";    //get the first score index 0
      cin >> score[0];  //only one score has been entered
      max = score[0];   //get 4 more scores
      for (i = 1; i < 5; i++)
      {
            cout << "Enter a score: ";
            cin >> score[i];  //check if the next score is bigger
            if (score[i] > max)
                  max = score[i]; //new highest score
      }
      //max will have the largest of the five numbers entered
      cout << "\nThe highest score is: " << max << endl
            << "The scores and their difference from max: " << endl;
      //loop through the entire array
      for (i = 0; i < 5; i++)
      {
            cout << score[i] << " off by "
                  << (max - score[i]) << endl << endl;
      }
      return 0;
}
```

ARRAYS

- Arrays are stored consecutive in memory

```
int a[6];
```

- Reserves enough memory for 6 integers
- The computer starts with `a[0]` address
- will add the correct number of bytes, in this case for an integer
- locate any other element (integer) in the array.
- There is no check for an illegal access (outside of the array)

ARRAYS

Initializing arrays:

```
//declaration and initialization of an array
int children[3] = { 2, 12, 1 };
//automatic size
int numbers[] = { 5, 12, 25 };
```

For loop examples:

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

int main()
{
	int numbers[] = { 5, 12, 25 }; //automatic size
	// for loop - 3 elements in the array
	for (int i = 0; i < 3; i++)

		//i is local to this for loop block
		cout << numbers[i] << endl;

	cout << endl;

	return 0;
}
```

ARRAYS IN FUNCTIONS

Indexed variable as an argument to a function is the same as any variable of that datatype

```
void MyFunction(int);

    int n = 10;
    int array[3] = { 2, 3, 4 };

    //sample function calls
    MyFunction(n); //integer variable
    MyFunction(array[1]);//indexed integer
    MyFunction(25);//literal value
```

FUNCTIONS WITH ARRAY PARAMETERS

```
#include <iostream>

using namespace std;

//this function may be used with arrays that are different sizes
//Precondition: count is the number of elements to be added to the array
//Postcondition: the array will be filed by the user with the (count) number of integers
void FillUp(int a[], int count);

//prints the integers in an array in a column
void PrintArray(const int a[], int count);
```

SEARCHING AN ARRAY

```
//returns the first location(index) the matches the target
int ArraySearch2(const int a[], int count, int target)
{
    for (int i = 0; i < count; i++)
    {
        if (a[i] == target)
            return i;
    }
    return -1;

}
```

2D ARRAYS

- The elements of an array can be arrays

```
int a[3][2]; //[row_size][column_size]
for(int i = 0; i < 3; i++)
   for(int j = 0; j < 2; j++)
          cout << “enter a number”;
          cin >> a[i][j];
```

- Elements are allocated contiguously in memory
- A can be thought of as a 3 X 2 matrix

2D ARRAYS

- The elements of an array can be arrays

```
int a[3][2]; //[row_size][column_size]
for(int i = 0; i < 3; i++)
{
   for(int j = 0; j < 2; j++)
   {
          cout << “enter a number”;
          cin >> a[i][j];
   }
{
```

- Elements are allocated contiguously in memory
- A can be thought of as a 3 X 2 matrix

POINTERS

POINTERS

pointer (pointer variable)

- a memory cell that stores the address of a data item
- syntax: *type *variable*

```
int  m  =  25;
int  *itemp;      /* a pointer to an integer */
```

POINTERS

- Must be assigned a specific data type to point to

```
FILE *inp; //identifier ready to point to a file
int *numptr; //identifier ready to point to an integer
char *letterptr; //identifier ready to point to a character
double *amtptr; //identifier ready to point to a double
```

- Declaring a pointer just provides an identifier (name) for the pointer but it **points to nothing.**
- To use the pointer we must point it to a variable location.
- The data types must match.

```
int *numPtr; //identifier ready to point to an integer
int number = 25;
numPtr = &number;//pointers must be initialized using & and the variable name
```

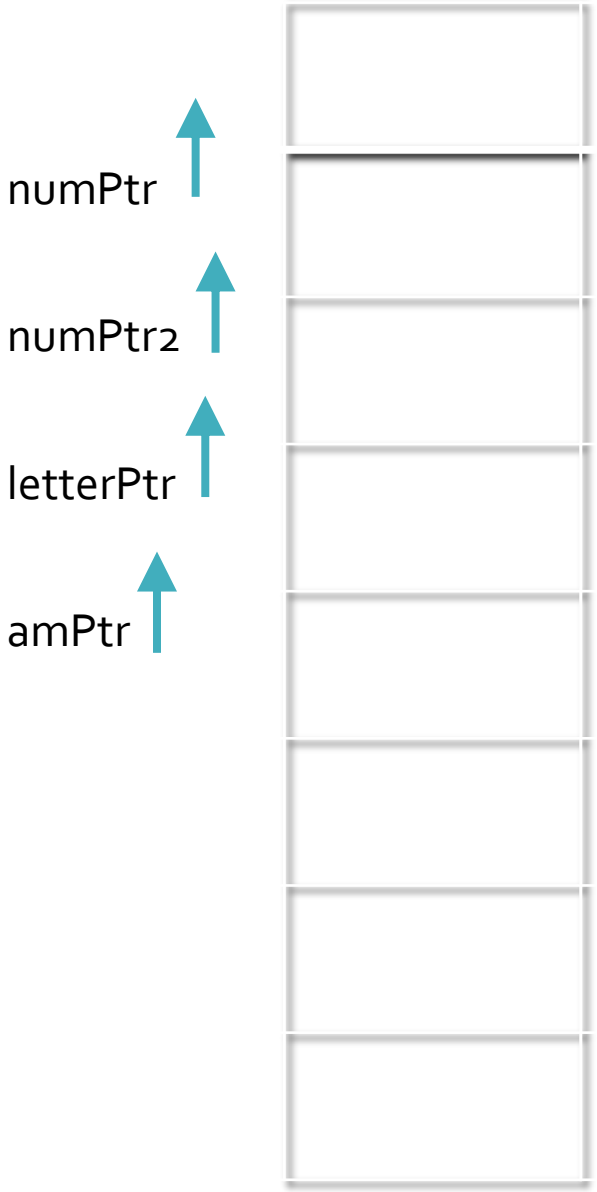
Pointer example Memory Map

```
int *numPtr; //identifier ready to point to an integer
int *numPtr2; //identifier ready to point to an integer

char *letterPtr; //identifier ready to point to a character
double *amtptr; //identifier ready to point to a double

int number = 25, number2 = 15;
numPtr = &number;//pointers must be initialized using & and the variable name

numPtr2 = &number2;//pointers must be initialized using & and the variable name

char letter = ‘X’;
letterPtr = &letter;//pointers must be initialized using & and the variable name

double amount = 55.5;
amtPtr = &amount;//pointers must be initialized using & and the variable name
```

Pointer example Memory Map

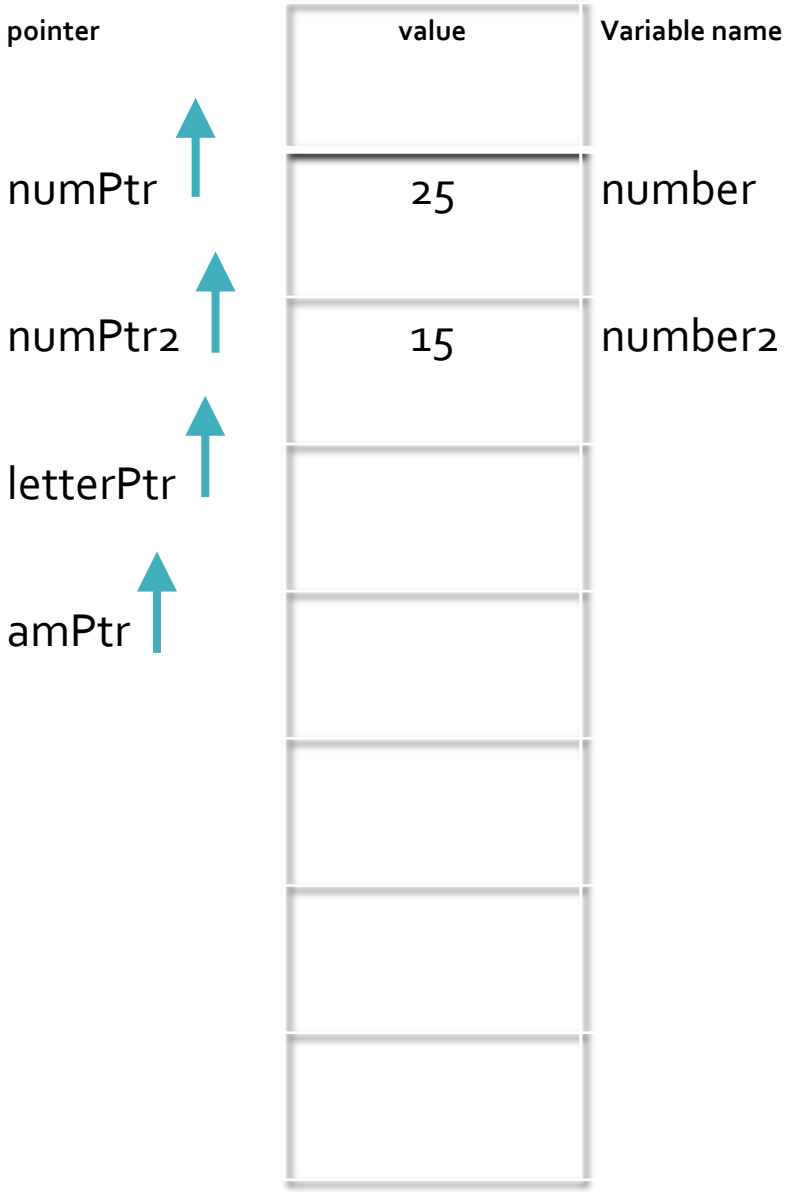
```
int *numPtr; //integer pointer
int *numPtr2; //integer pointer

char *letterPtr; //character pointer
double *amtptr; //double pointer
```

NOTE: The pointers are declared but they do not point to anything valid yet



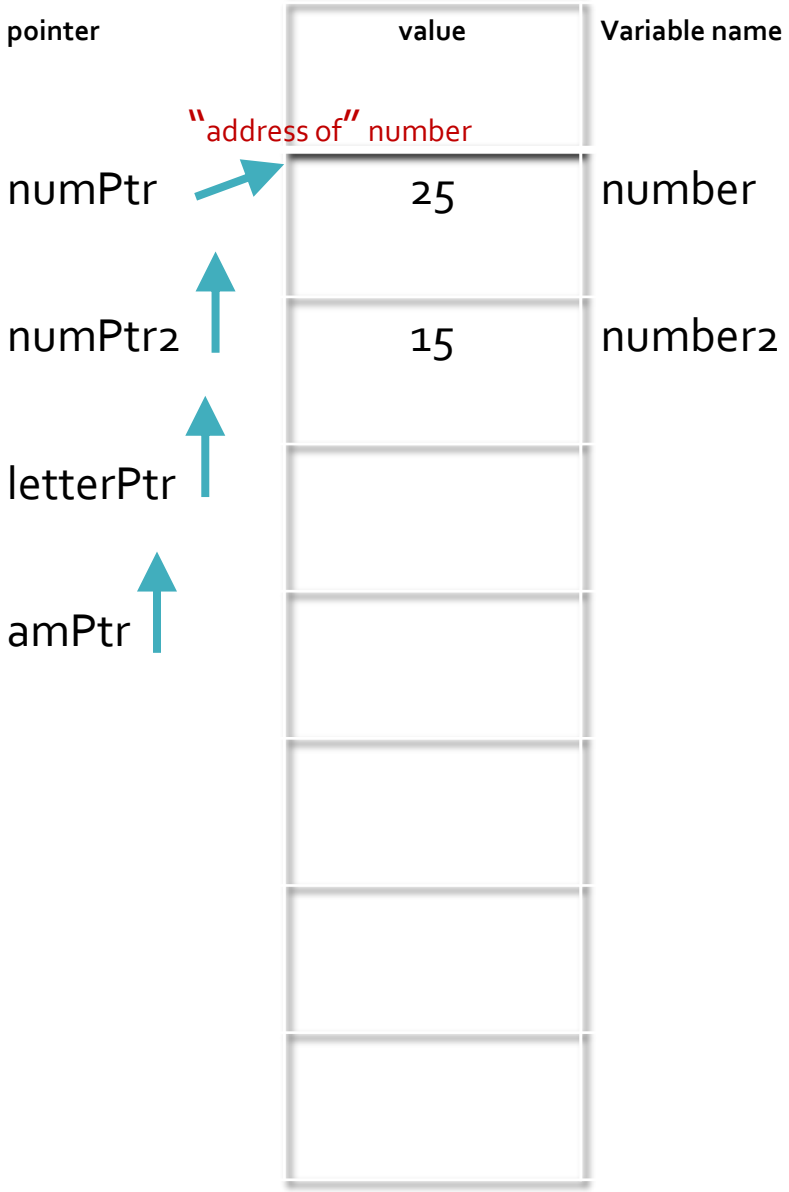
Pointer example Memory Map

```
int *numPtr; //integer pointer
int *numPtr2; //integer pointer

char *letterPtr; //character pointer
double *amtptr; //double pointer

NOTE: The pointers are declared but they
do not point to anything valid yet

int number = 25, number2 = 15;
```



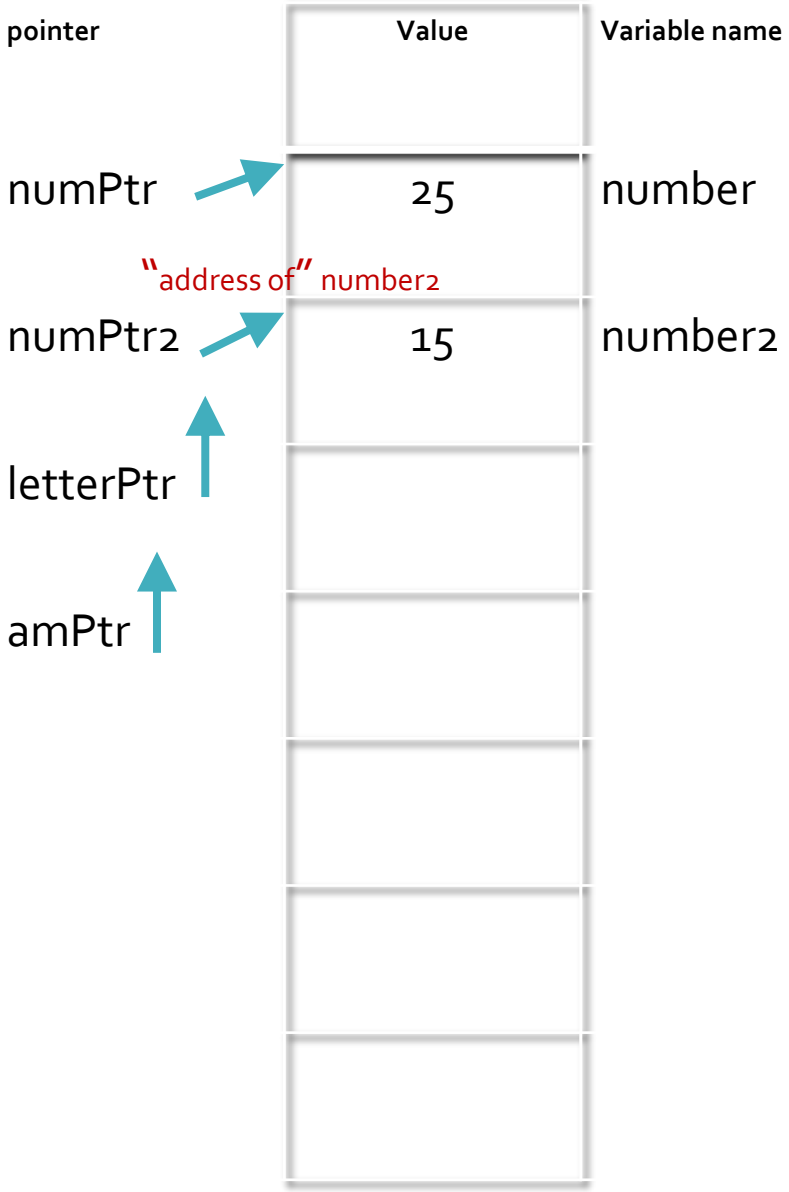
Pointer example Memory Map

```
int *numPtr; //integer pointer
int *numPtr2; //integer pointer

char *letterPtr; //character pointer
double *amtptr; //double pointer


int number = 25, number2 = 15;
numPtr = &number; //numPtr = “address of” number
```



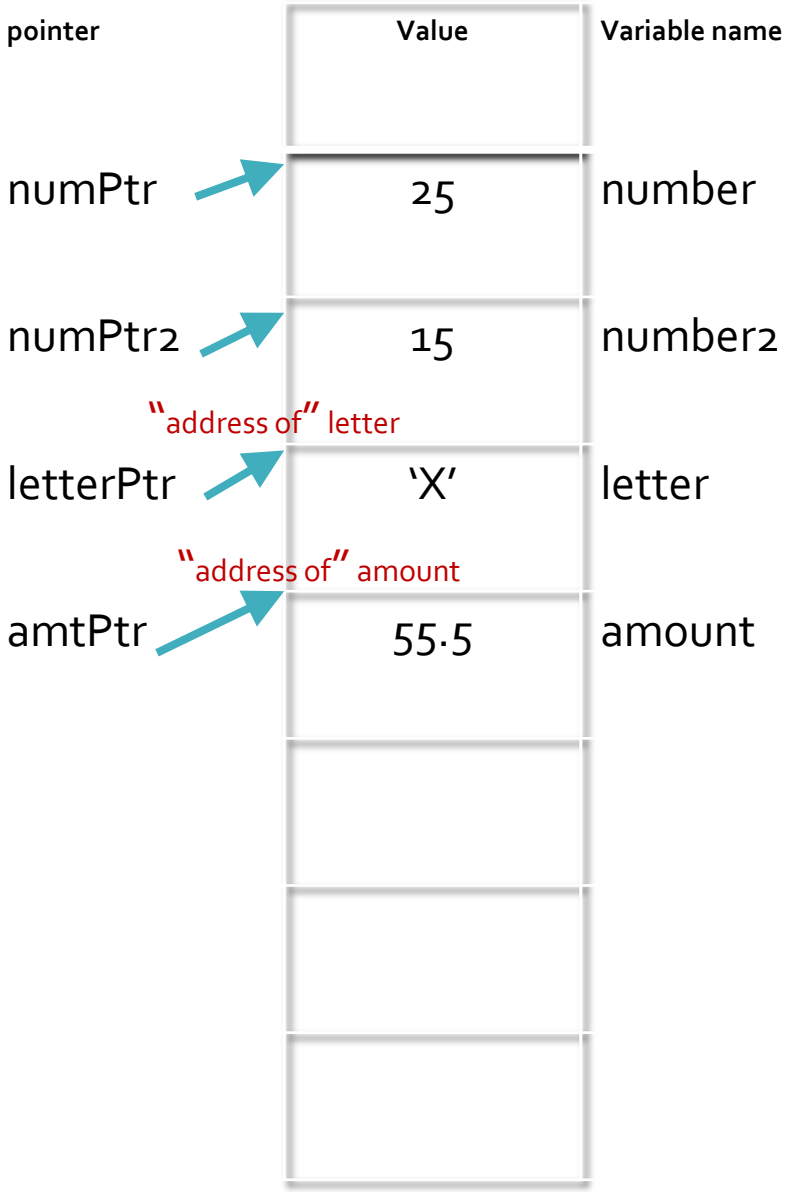
Pointer example Memory Map

```
int *numPtr; //integer pointer
int *numPtr2; //integer pointer

char *letterPtr; //character pointer
double *amtptr; //double pointer


int number = 25, number2 = 15;
numPtr = &number; //numPtr = “address of” number
numPtr2 = &number2; //numPtr = “address of” number2
```



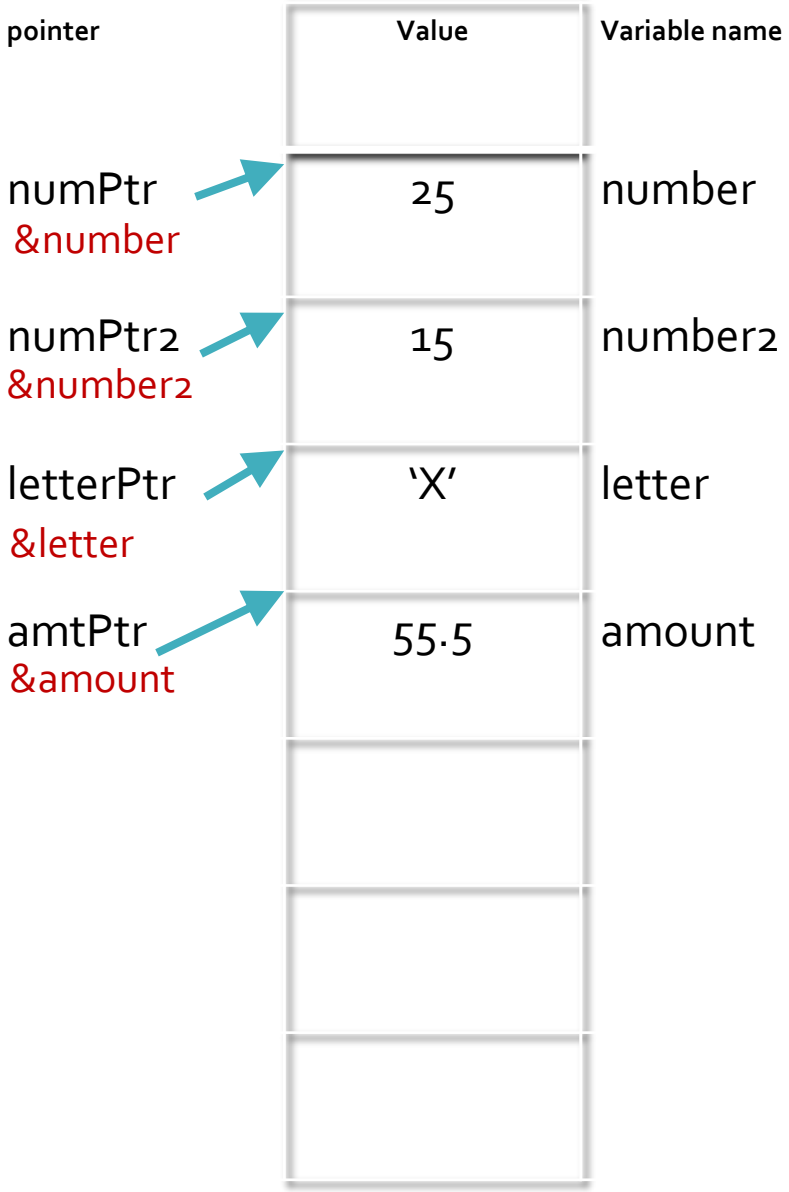
Pointer example Memory Map

```
int *numPtr; //integer pointer
int *numPtr2; //integer pointer

char *letterPtr; //character pointer
double *amtptr; //double pointer


int number = 25, number2 = 15;
numPtr = &number; //numPtr = "address of" number
numPtr2 = &number2; //numPtr = "address of" number2


char letter = 'X';
letterPtr = &letter;//letterPtr = "address of" letter

double amount = 55.5;
amtPtr = &amount;//amtPtr = "address of" amount
```



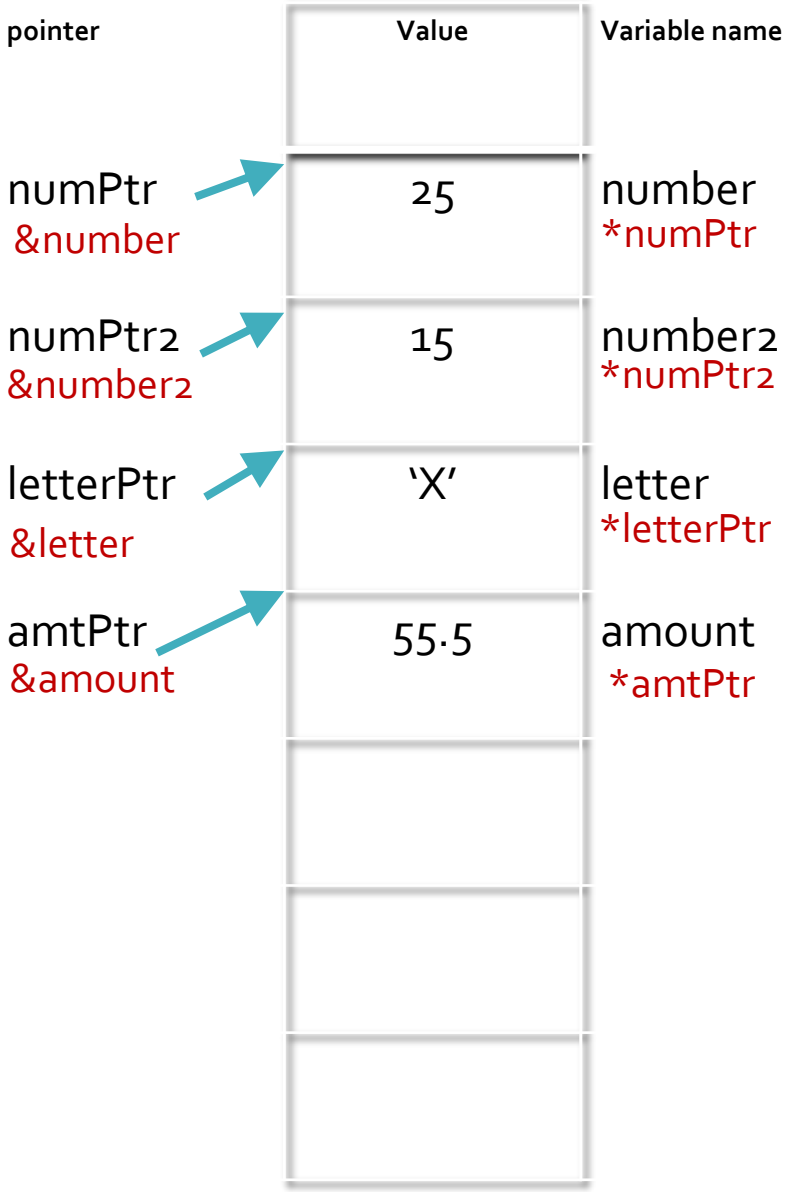
Pointer example Memory Map

```
int *numPtr; //integer pointer
int *numPtr2; //integer pointer

char *letterPtr; //character pointer
double *amtptr; //double pointer


int number = 25, number2 = 15;
numPtr = &number; //numPtr = “address of” number
numPtr2 = &number2; //numPtr = “address of” number2


char letter = ‘X’;
letterPtr = &letter;//letterPtr = “address of” letter

double amount = 55.5;
amtPtr = &amount;//amtPtr = “address of” amount
```



Pointer example Memory Map

```
int *numPtr; //integer pointer
int *numPtr2; //integer pointer

char *letterPtr; //character pointer
double *amtptr; //double pointer


int number = 25, number2 = 15;
numPtr = &number; //numPtr = “address of” number
numPtr2 = &number2; //numPtr = “address of” number2


char letter = ‘X’;
letterPtr = &letter;//letterPtr = “address of” letter

double amount = 55.5;
amtPtr = &amount;//amtPtr = “address of” amount
```



Pointer example Memory Map

```
int *numPtr; //integer pointer
int *numPtr2; //integer pointer

char *letterPtr; //character pointer
double *amtptr; //double pointer


int number = 25, number2 = 15;
numPtr = &number; //numPtr = “address of” number
numPtr2 = &number2; //numPtr = “address of” number2


char letter = ‘X’;
letterPtr = &letter;//letterPtr = “address of” letter

double amount = 55.5;
amtPtr = &amount;//amtPtr = “address of” amount
```