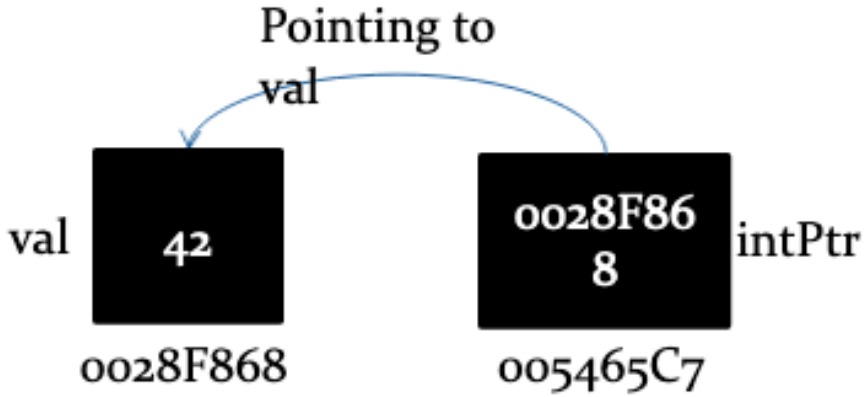
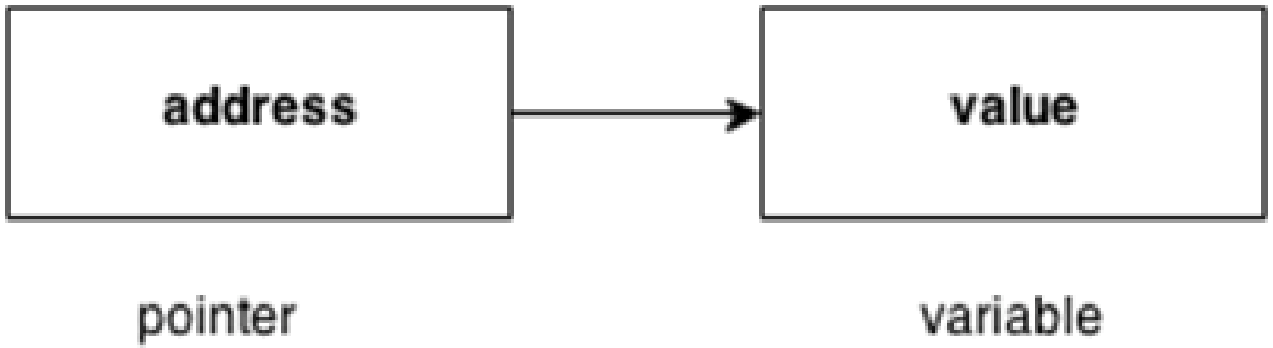
Pointers in C++

Overview of Pointers



- The pointer in C++ language is a variable.
- It is also known as locator or indicator that points to an address of a value.



- A Pointer in C++ is variable whose value is a memory address.
- With pointers many memory locations can be referenced.
- Some data structures use pointers (e.g. linked list, tree).

The * and & operators

- & operator is the address operator
- * operator is the dereferencing operator.
  - It is used in pointers declaration.

Pointer Declaration

- Pointers are declared as follows:

<type> * variable_name ;

e.g.

```
int  * xPtr;    // xPtr is a pointer to data of type integer

char * cPtr;  //cPtr is a pointer to data of type character

void * yPtr;    // yPtr is a generic pointer, represents any type
```

Pointer Assignment

- Assignment can be applied on pointers of the same type
- If not the same type, a cast operator must be used
- Exception: pointer to **void** does not need casting to convert a pointer to **void** type
- **void** pointers cannot be dereferenced
- **Example**

```
int *xPtr, *yPtr;
int x = 5;
...
xPtr = &x;   // xPtr now points to address of  x
yPtr = xPtr;  // now yPtr and xPtr point to x
```

Examples on Pointers (Contd.)

```
//A program to test pointers and references
#include <iostream.h>
void main ( )
{   int intVar = 10;
    int  *intPtr;    // intPtr is a pointer
    intPtr = & intVar;
    cout << "\nLocation of intVar: " << & intVar;
    cout << "\nContents of intVar: " << intVar;
    cout << "\nLocation of intPtr: " << & intPtr;
    cout << "\nContents of intPtr: " << intPtr;
    cout << "\nThe value that intPtr points to: " << * intPtr;
}
```

OUTPUT

```
Location of intVar: 0x7fff92a6ab4c
Contents of intVar: 10
Location of intPtr: 0x7fff92a6ab40
Contents of intPtr: 0x7fff92a6ab4c T
he value that intPtr points to: 10
```

NULL Pointer

- It is always a good practice to assign the pointer NULL to a pointer variable in case you do not have exact address to be assigned.
- A pointer that is assigned NULL is called a **null** pointer.
- The NULL pointer is a constant with a value of zero (0).
- **Example**

```
#include <iostream>
using namespace std;
int main () {
    int *ptr = NULL;
    // The standard keyword for NULL pointer in C++ 11 can be declared as: int *ptr = nullptr
    cout << "The value of ptr is " << ptr ;
    return 0;
}
```

Output

```
The value of ptr is 0
```

Pointer Arithmetic

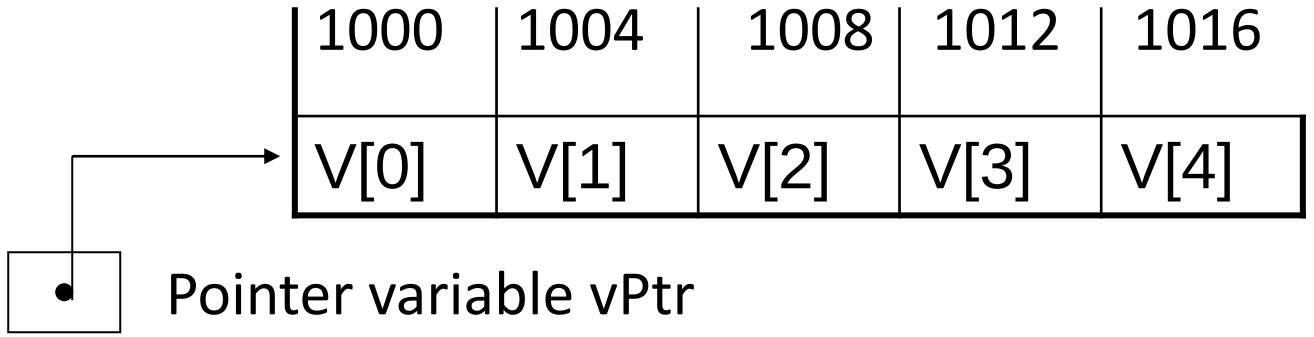
- Increment / decrement pointers (++ or --)
- Add / subtract an integer to/from a pointer

  (+ or += , - or -=)
- Pointers may be subtracted from each other
- Pointer arithmetic is meaningless unless performed on an array.

Pointer Arithmetic (Contd.)

- **Example**

  Consider an integer array of 5 elements on a machine using 4 bytes for integers.



- vPtr pointes to first element V[0] (location 1000)

  i.e. vPtr = 1000

- vPtr +=2; sets vPtr to 1008

  i.e. vPtr points to V[2]

Pointer Arithmetic (Contd.)

- **Subtracting pointers**
  - Returns the number of elements between two addresses

e.g. if v is an array and

vPtr1 = v[0];

vPtr2 = v[2];

then vPtr2 – vPtr1 = 2 (i.e. 2 addresses)

Arrays in Pointer

- Pointers and arrays are strongly related.
- A pointer that points to the beginning of an array can access that array by using either pointer arithmetic or array-style indexing.

Array in Pointer (Example)

```
#include <iostream>
using namespace std;
const int MAX = 3;
int main () {
      int var[MAX] = {10, 100, 200};
      int *ptr;
      ptr = var; // let us have array address in pointer.
      for (int i = 0; i < MAX; i++) {
      cout << "Address of var[" << i << "] = ";
      cout << ptr << endl; cout << "Value of var[" << i << "] = ";
      cout << *ptr << endl; // point to the next location
      ptr++;
      }
      return 0;
}
```

OUTPUT

```
Address of var[0] = 0x7ffca43c75a8
Value of var[0] = 10
Address of var[1] = 0x7ffca43c75ac
Value of var[1] = 100
Address of var[2] = 0x7ffca43c75b0
Value of var[2] = 200
```

Pointers in function

- C++ allows you to pass a pointer to a function.
- To do so, simply declare the function parameter as a pointer type.
- Following a simple example where we pass an unsigned long pointer to a function and change the value inside the function which reflects back in the calling function

Pointers in function (Contd.)

- **Example**
- // function definition to swap the values.
- void swap(int *x, int *y) {
  - int temp;
  - temp = *x; /* save the value at address x */
  - *x = *y; /* put y into x */
  - *y = temp; /* put x into y */
  - return;
- }

Example (Contd.)

OUTPUT

```
Before swap, value of a :100
Before swap, value of b :200
After swap, value of a :200
After swap, value of b :100
```

```
#include <iostream>
using namespace std;
void swap(int *x, int *y); // function declaration
int main () {
    int a = 100; int b = 200; // local variable declaration:
    cout << "Before swap, value of a :" << a << endl;
    cout << "Before swap, value of b :" << b << endl;
    swap(&a, &b);
    /* calling a function to swap the values. * &a indicates pointer to a ie. address of variable a and
    * &b indicates pointer to b ie. address of variable b. */
    cout << "After swap, value of a :" << a << endl;
    cout << "After swap, value of b :" << b << endl;
    return 0;
}
```

Relations Between Pointers and Arrays

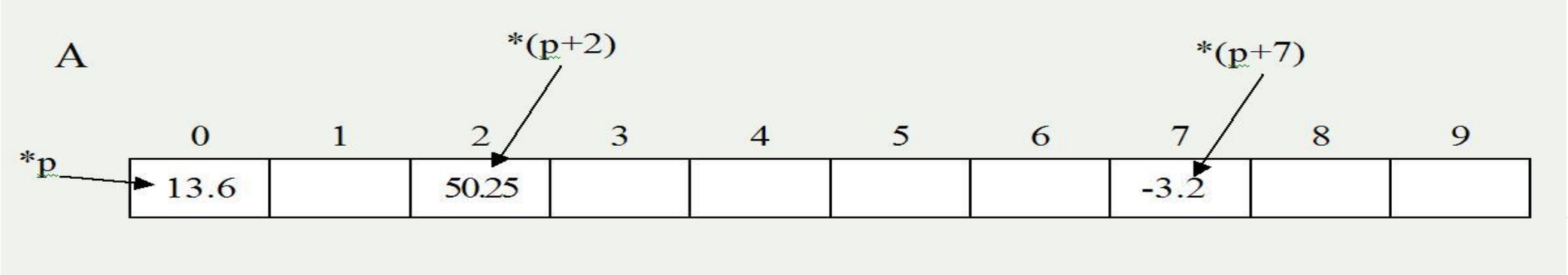
- Arrays and pointers are closely related.
  - Array name is like constant pointer
  - Pointers can do array subscribing operations
  - If we declare an array A[4] and a pointer aPtr
    - aPtr is equal to A

      aPtr == A
    - aPtr is equal to the address of the first element of A

      aPtr == & A[0]

Relations Between Pointers and Arrays (Cont.)

- Accessing array elements with pointers:
- Element A[i] can be accessed by *(aPtr+i)
  - This is called pointer/offset notation
- Array itself can use pointer arithmetic
  - A[2] is same as *(A+2)
- Pointers can be subscripted (i.e. pointer/subscript notation)
  - aPtr [2] is same as A[2]



Accessing Arrays using pointers

- In an array declaration, the array name is a constant pointer to the first element of the array. e.g. **double balance[50];**
- **balance** is a pointer to **&balance[0]**, which is the address of the first element of the array balance.
- Therefore, in the program below, **p** points to the address of first element of the array, i.e **&balance[0]**.

```
double *p;
double balance[10];
p = balance;
```

Accessing Arrays using pointers (Contd.)

- You can access subsequent elements in the array with pointers. e.g. ***(balance + 8)** returns the data at **balance[8]**.
- Once you store the address of first element in **p**, you can access subsequent array elements using ***p, *(p+1), *(p+2)** and so on.
- An example to demonstrate this can be seen in the next slide.

Accessing Arrays using pointers (Contd.)
Example

```
#include <iostream>

using namespace std;

int main () {
      double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
      double *p;
      p = balance;
      cout << "Array values using pointer " << endl; // output each array element's value
      for ( int i = 0; i < 5; i++ ) {
            cout << "*(p + " << i << ") : ";
            cout << *(p + i) << endl;
      }
       cout << "Array values using balance as address " << endl;
      for ( int i = 0; i < 5; i++ ) {
            cout << "*(balance + " << i << ") : ";
            cout << *(balance + i) << endl;
      }
      return 0;
}
```

OUTPUT

```
Array values using pointer
*(p + 0) : 1000
*(p + 1) : 2
*(p + 2) : 3.4
*(p + 3) : 17
*(p + 4) : 50
Array values using balance as address
*(balance + 0) : 1000
*(balance + 1) : 2
*(balance + 2) : 3.4
*(balance + 3) : 17
*(balance + 4) : 50
```

Arrays of Pointers

```
#include <iostream>
using namespace std;
int main()  {
    int *ptr;  // integer pointer declaration
    int marks[10];  // marks array declaration
    std::cout << "Enter the elements of an array :" << std::endl;
    for(int i=0;i<10;i++)
    {
        cin>>marks[i];
    }
    ptr=marks; // both marks and ptr pointing to the same element..
    std::cout << "The value of *ptr is :" <<*ptr<< std::endl;
    std::cout << "The value of *marks is :" <<*marks<<std::endl;
}
```

OUTPUT

```
Enter the elements of an array:
1
2
3
4
5
6
7
8
9
10
The value of *ptr is :1
The value of *marks is :1
```

Array of Pointers

- An array of pointers is an array that consists of variables of pointer type, which means that the variable is a pointer addressing to some other element.
- Declaring an array of pointer holding 5 integer pointers;
  - **int *ptr[5];** // array of 5 integer pointer.
- Element of an array of a pointer can also be initialized by assigning the address of some other element.
- Let's observe this case through an example.
- **int a;** // variable declaration.
- **ptr[2] = &a;** //assigning the address of **'a'** to the 3rd element of array 'ptr'.
- ***ptr[2];** // retrieve the value of **'a'** by dereferencing the pointer.

Array of Pointer (Example)

```
#include <iostream>
using namespace std;
int main()
{
   int ptr1[5]; // integer array declaration
   int *ptr2[5]; // integer array of pointer declaration
   std::cout << "Enter five numbers :" << std::endl;
   for(int i=0;i<5;i++)
   {
      std::cin >> ptr1[i];
   }
   for(int i=0;i<5;i++)
   {
      ptr2[i]=&ptr1[i];
   }
    std::cout << "The values are" << std::endl; // printing the values of ptr1 array
   for(int i=0;i<5;i++)
   {
      std::cout << *ptr2[i] << std::endl;
   }
}
```

Array of Pointers to Strings

- An array of pointer to strings is an array of character pointers that holds the address of the first character of a string or we can say the base address of a string.
- It can be declared as follow;
- **char *names[5] = {"john", "Peter", "Marco", "Devin", "Ronan"};**
- OR
- **char *names[] = {"john", "Peter", "Marco", "Devin", "Ronan"};**

Example

```
#include <iostream>
using namespace std;
int main(){
    char name[ ]= "Sam";
    char *p;
    p = name; /* for string, only this declaration will store its base address */
    while( *p != '\0') {
        cout << *p; p++;
    }
    return 0;
}
```

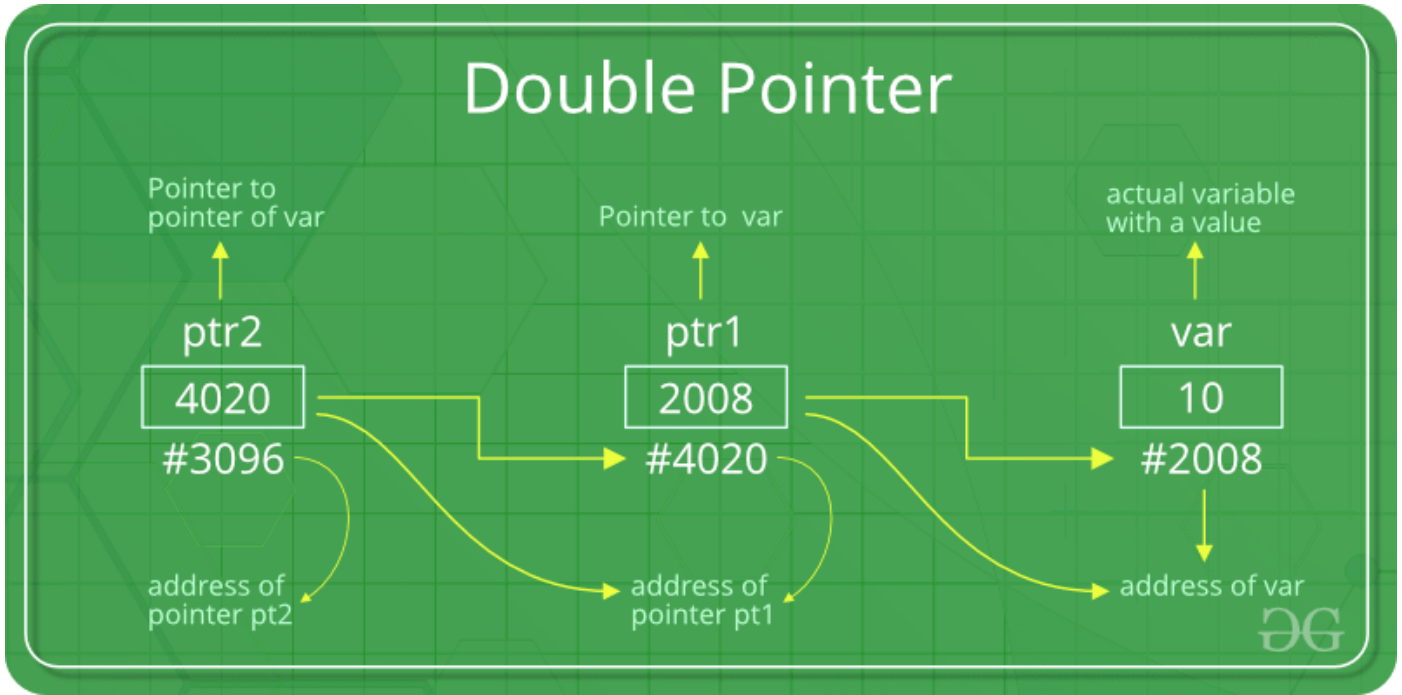
In this example, since **p** stores the address of name[0], therefore the value of ***p** equals the value of name[0] i.e., **'S'**. So in while loop, the first character gets printed and **p++** increases the value of **p** by 1 so that now p+1 points to name[1]. This continues until the pointer reaches the end of the string i.e., before *p becomes equal to '\0'.

OUTPUT

Sam

Pointer to Pointer

- A pointer to a pointer is a form of multiple indirection or a chain of pointers.
- A variable that is a pointer to a pointer must be declared as such.
- This is done by placing an additional asterisk in front of its name. e.g. **int **var;**



Pointer to a Pointer (Example)

```
#include <iostream>
using namespace std;
int main () {
      int var; int *ptr; int **pptr;
      var = 3000; // take the address of var
      ptr = &var; // take the address of ptr using address of operator &
      pptr = &ptr; // take the value using pptr
      cout << "Value of var :" << var << endl;
      cout << "Value available at *ptr :" << *ptr << endl;
      cout << "Value available at **pptr :" << **pptr << endl;
      return 0;
}
```

OUTPUT

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