

Introduction to Pointers

- A Pointer is a variable which holds the address of another variable of same data type.
- Pointers are used to access memory and manipulate the address.

Address of Operator (&)

- Whenever a variable is defined , a memory location is assigned for it, in which it's value will be stored. We can easily check this memory address, using the & symbol.
- If var is the name of the variable, then &var will give it's address.

```
#include <iostream>
```

```
using namespace std;
```

```
int main () {
```

```
    int var1;
```

```
    char var2[10];
```

```
    cout << "Address of var1 variable: ";
```

```
    cout << &var1 << endl;
```

```
    cout << "Address of var2 variable: ";
```

```
    cout << &var2 << endl;
```

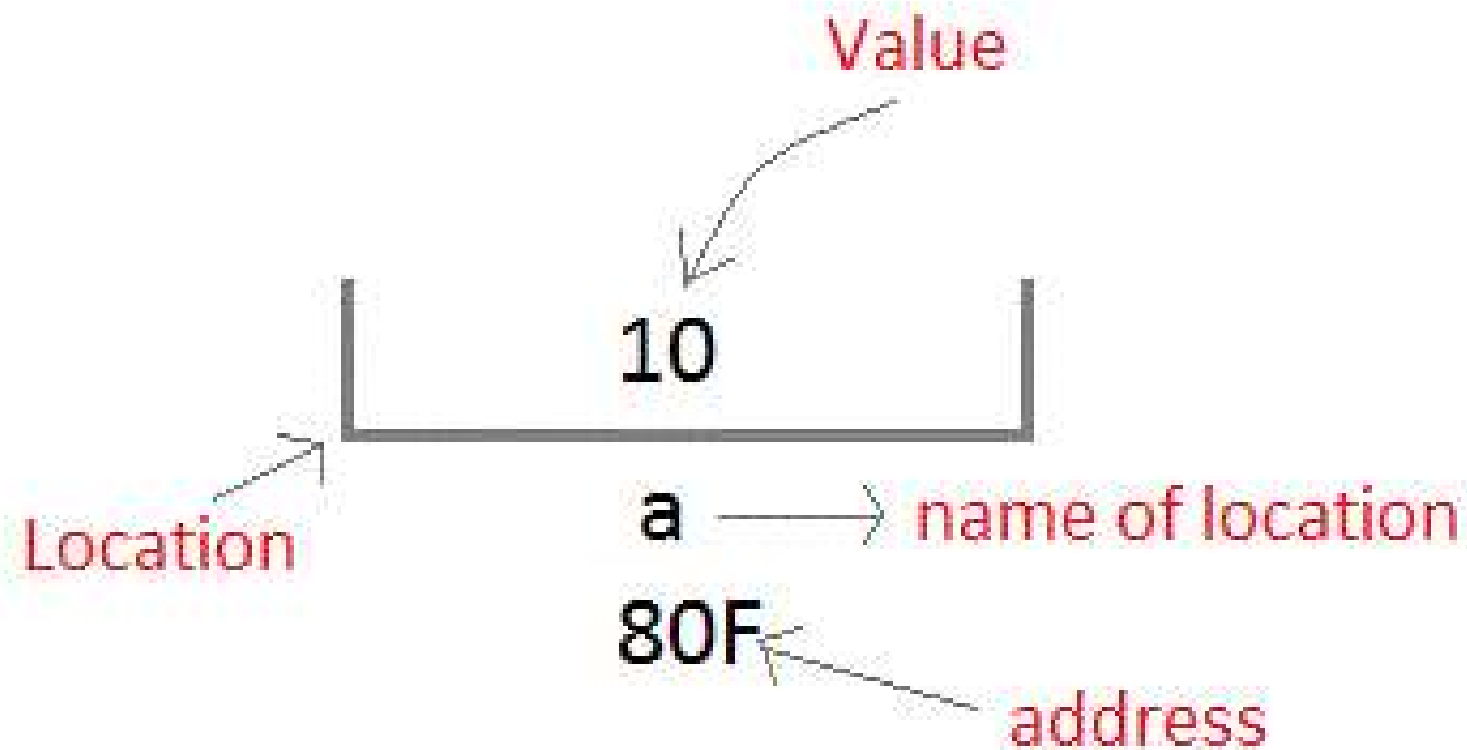
```
    return 0;
```

```
}
```

- Address of var1 variable: 0xbfebd5c0
- Address of var2 variable: 0xbfebd5b6

Dereference Operator (*)

- Whenever a variable is declared in a program, system allocates a location i.e an address to that variable in the memory, to hold the assigned value. This location has its own address number, which we just saw above.
- Let us assume that system has allocated memory location 80F for a variable a.
- `int a = 10;`
- storage of variable in C



- We can access the value 10 either by using the variable name a or by using its address 80F.
- The question is how we can access a variable using its address? Since the memory addresses are also just numbers, they can also be assigned to some other variable. The variables which are used to hold memory addresses are called Pointer variables.
- A pointer variable is therefore nothing but a variable which holds an address of some other variable. And the value of a pointer variable gets stored in another memory location.

Benefits of using pointers

Below we have listed a few benefits of using pointers:

Pointers are more efficient in handling Arrays and Structures.

Pointers allow references to function and thereby helps in passing of function as arguments to other functions.

It reduces length of the program and its execution time as well.

It allows C language to support Dynamic Memory management.

Declaring Pointers

syntax

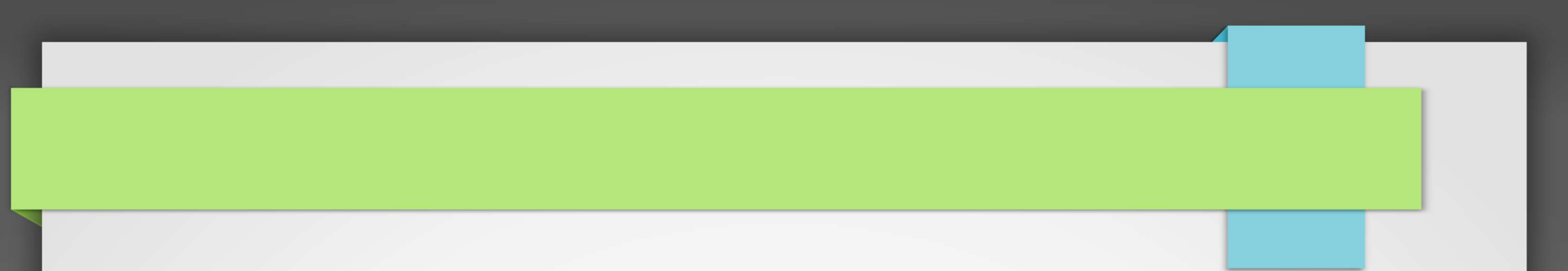
- `datatype *pointer_name;`

Examples:

- `int *ip // pointer to integer variable`
- `float *fp; // pointer to float variable`
- `double *dp; // pointer to double variable`
- `char *cp; // pointer to char variable`

Initialization of Pointer variable

- Pointer Initialization is the process of assigning address of a variable to a pointer variable.
- Pointer variable can only contain address of a variable of the same data type.
- In C language address operator & is used to determine the address of a variable.
- The & (immediately preceding a variable name) returns the address of the variable associated with it.



```
#include<iostream.h>
```

```
void main()
```

```
{
```

```
    int a = 10;
```

```
    int *ptr;    //pointer declaration
```

```
    ptr = &a;    //pointer initialization
```

```
}
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main () {
```

```
    int var = 20; // actual variable declaration.
```

```
    int *ip;      // pointer variable
```

```
    ip = &var;    // store address of var in pointer variable
```

```
    cout << "Value of var variable: ";
```

```
    cout << var << endl;
```

```
// print the address stored in ip pointer variable
```

```
    cout << "Address stored in ip variable: ";
```

```
    cout << ip << endl;
```

```
// access the value at the address available in pointer
```

```
    cout << "Value of *ip variable: ";
```

```
    cout << *ip << endl;
```

```
    return 0;
```

```
}
```

Pointers and Functions

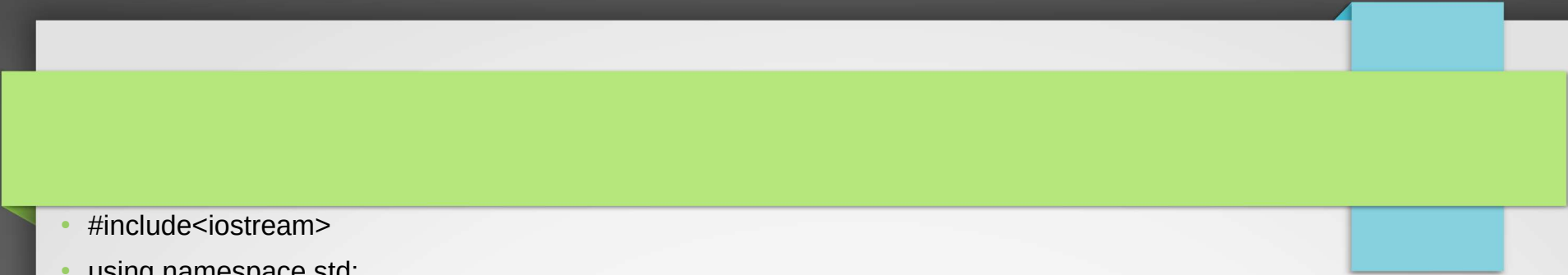
- Pointer as a function parameter is used to hold addresses of arguments passed during function call.
- This is also known as call by reference.
- When a function is called by reference any change made to the reference variable will effect the original variable.

Passing Pointers as Arguments to Functions

```
void swap(int *a, int *b);
int main()
{
    int m = 10, n = 20;
    cout<<"before Swapping:\n\n";
    cout<<m;
    cout<<n;
    swap(&m, &n);    //passing address of m and n to the swap function
    cout<<"After Swapping:\n\n";
    cout<<m;
    cout<<n;
    return 0;
}
void swap(int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

Returning Pointer from a function

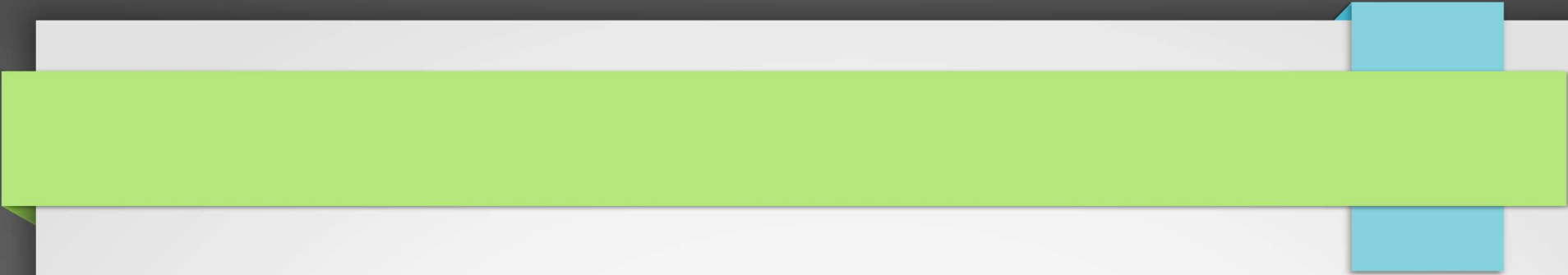
- A function can also return a pointer to the calling function. In this case you must be careful, because local variables of function doesn't live outside the function.
- They have scope only inside the function. Hence if you return a pointer connected to a local variable, that pointer will be pointing to nothing when the function ends.



- `#include<iostream>`
- `using namespace std;`
- `int* larger(int*, int*);`
-
- `int main()`
- `{`
- `int a = 15;`
- `int b = 92;`
- `int *p;`
- `p = larger(&a, &b);`
- `cout<<*p<<" is larger";`
- `return 0;`
- `}`
-
- `int* larger(int *x, int *y)`
- `{`
- `if(*x > *y)`
- `return x;`
- `else`
- `return y;`
- `}`

Pointer to functions

- It is possible to declare a pointer pointing to a function which can then be used as an argument in another function.
- A pointer to a function is declared as follows,
- `type (*pointer-name)(parameter);`
- Here is an example :
- `int (*sum)();` //legal declaration of pointer to function
- `int *sum();` //This is not a declaration of pointer to function



```
int sum(int, int);
```

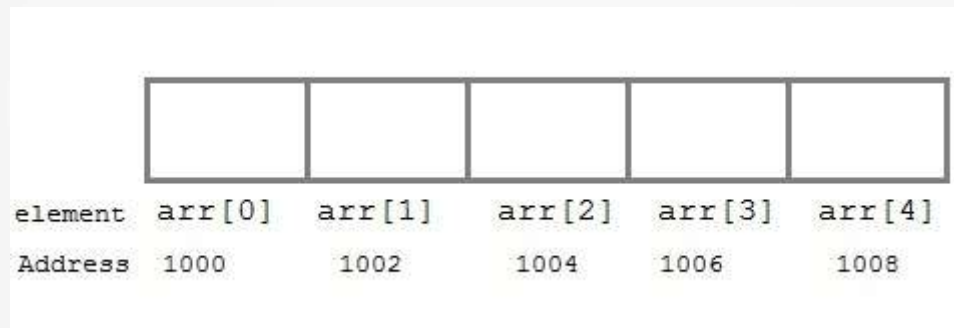
```
int (*s)(int, int);
```

```
s = sum;
```

Pointer and Arrays

- When an array is declared, compiler allocates sufficient amount of memory to contain all the elements of the array.
- Base address i.e address of the first element of the array is also allocated by the compiler.
- Suppose we declare an array arr,
- `int arr[5] = { 1, 2, 3, 4, 5 };`

-
-
-
- Assuming that the base address of arr is 1000 and each integer requires two bytes, the five elements will be stored as follows:
- address of array



- Here variable arr will give the base address, which is a constant pointer pointing to the first element of the array, `arr[0]`. Hence arr contains the address of `arr[0]` i.e 1000. In short, arr has two purpose - it is the name of the array and it acts as a pointer pointing towards the first element in the array.
- arr is equal to `&arr[0]` by default

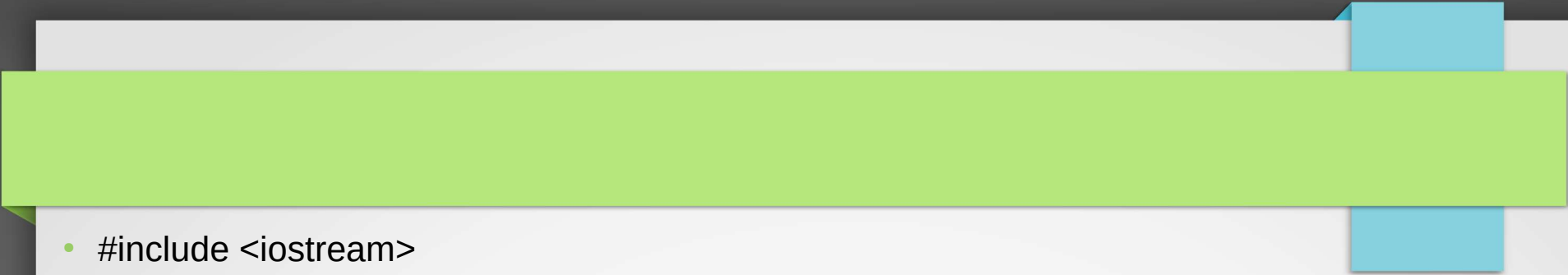


```
int *p;
```

```
p = arr;
```

```
or,
```

```
p = &arr[0];
```



- `#include <iostream>`
- `using namespace std;`
-
- `int main()`
- `{`
- `int i;`
- `int a[5] = {1, 2, 3, 4, 5};`
- `int *p = a; // same as int*p = &a[0]`
- `for (i = 0; i < 5; i++)`
- `{`
- `cout<<*p<<endl;`
- `p++;`
- `}`
-
- `return 0;`
- `}`

Pointer to Multidimensional Array

- A multidimensional array is of form, $a[i][j]$. Lets see how we can make a pointer point to such an array. As we know now, name of the array gives its base address. In $a[i][j]$, a will give the base address of this array, even $a + 0 + 0$ will also give the base address, that is the address of $a[0][0]$ element.
- Here is the generalized form for using pointer with multidimensional arrays.
- $*(*(a + i) + j)$
- which is same as,
- $a[i][j]$

Pointer and Character strings

- Pointer can also be used to create strings. Pointer variables of char type are treated as string.

```
char *str = "Hello";
```

- -----
- char *str;
- str = "hello";
- puts(str);