GE3 Computer Science

C and C ++ Lecture series *for*

B.SC 3rd semester *by*

Subhadip Mukherjee

Department of computer science

Kharagpur College

LECTURE 13

Introduction to C++ Programming

- What is programming?

Programming is taking

A ***problem***

Find the area of a rectangle

A set of ***data***

length

width

A set of ***functions***

area = length * width

Then,

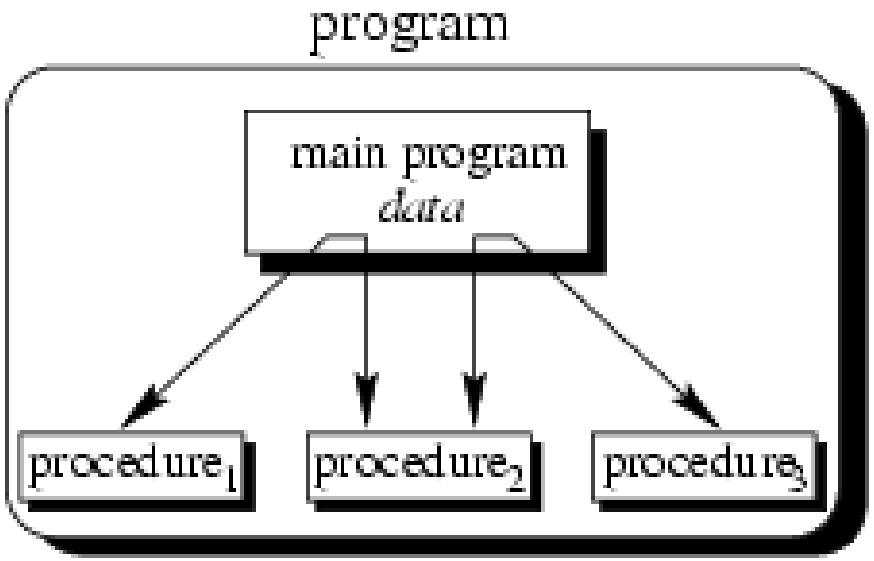
Applying functions to data to solve the problem

Introduction to C++ Programming

Programming Concept Evolution

- Procedural
- Object-Oriented

Procedural Concept



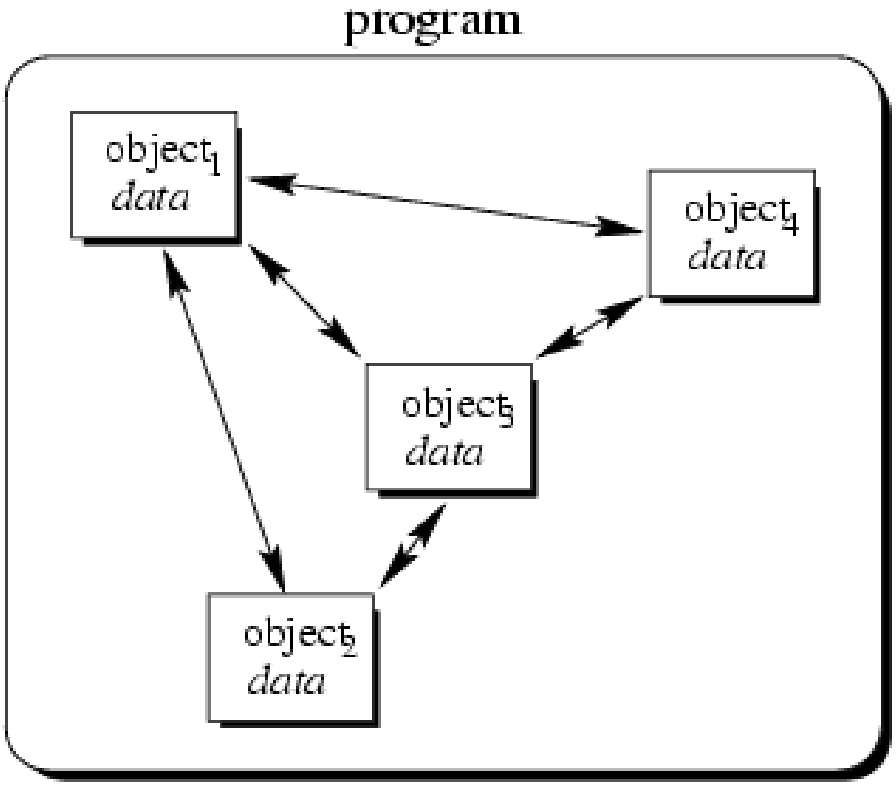
- The main program coordinates calls to procedures and hands over appropriate data as parameters.

Procedural Concept (II)

- Procedural Languages
  - C, Pascal, Basic, Fortran
  - Facilities to
    - Pass arguments to functions
    - Return values from functions
- For the rectangle problem, we develop a function

```
int compute_area (int l, int w){
    return ( l * w );
}
```

Object-Oriented Concept



- Objects of the program interact by sending messages to each other

Characteristics of OOPL

- **Encapsulation:** combining data structure with actions
  - Data structure: represents the properties, the state, or characteristics of objects
  - Actions: permissible behaviors that are controlled through the member functions

  Data abstraction: Process of making certain data inaccessible
- **Inheritance:** Ability to derive new objects from old ones
  - permits objects of a more specific class to inherit the properties (data) and behaviors (functions) of a more general/base class
  - ability to define a hierarchical relationship between objects
- **Polymorphism:** Ability for different objects to interpret functions differently

Basic C++ Extension from C

- comments

```
/* You can still use the old comment style, */
/* but you must be // very careful about mixing them */
// It's best to use this style for 1 line or partial lines
/* And use this style when your comment
 consists of multiple lines */
```

- cin and cout (and #include <iostream.h>)

```
cout << "hey";
char name[10];
cin >> name;
cout << "Hey " << name << ", nice name." << endl;
cout << endl;      // print a blank line
```

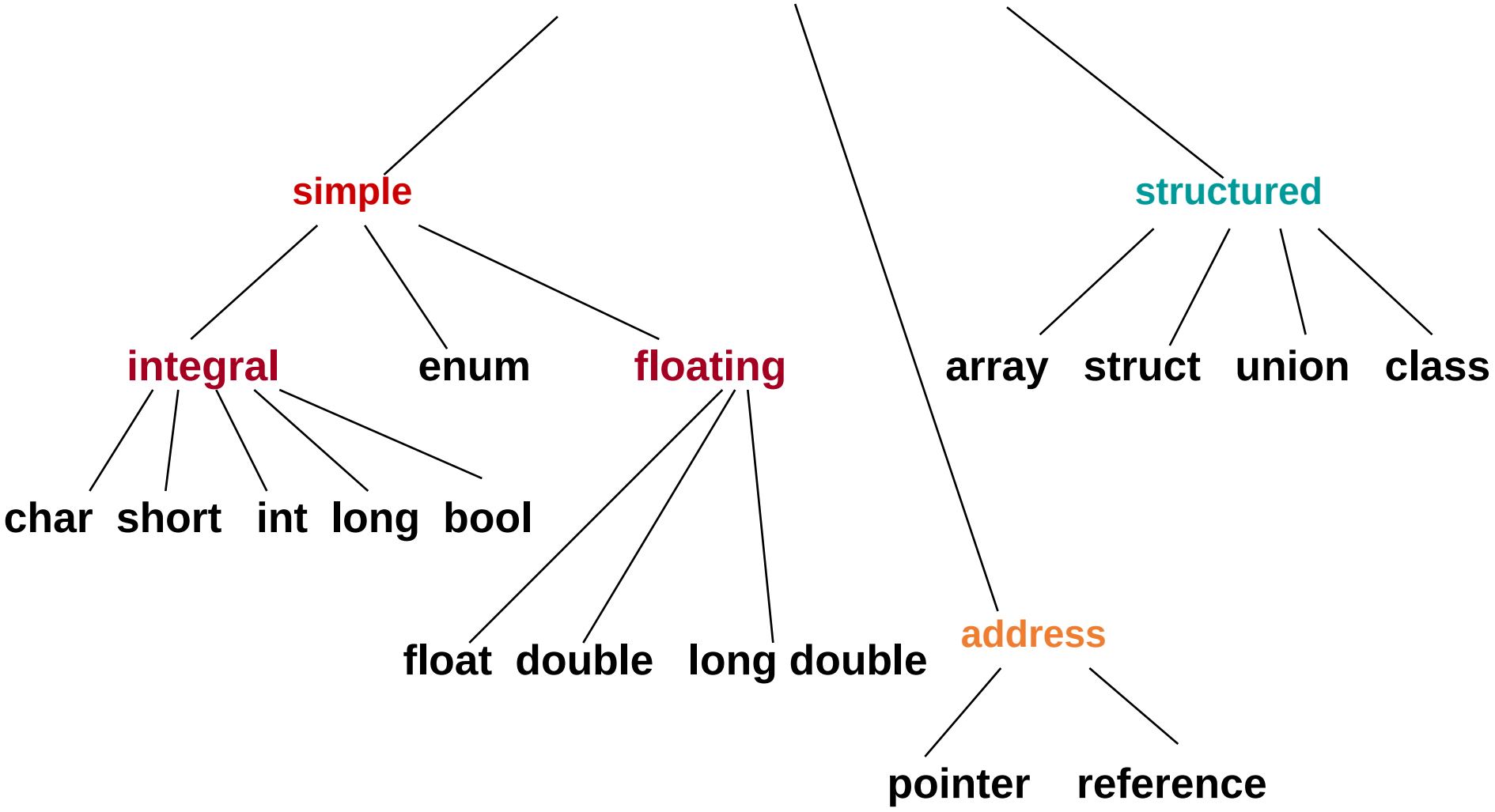
- declaring variables almost anywhere

```
// declare a variable when you need it
   for (int k = 1; k < 5; k++){
       cout << k;
  }
```

Properties C++

- Is a better C
- Expressive
- Supports Data Abstraction
- Supports OOP
- Supports Generic Programming
  - Containers
    - Stack of char, int, double etc
  - Generic Algorithms
    - sort(), copy(), search() any container Stack/Vector/List

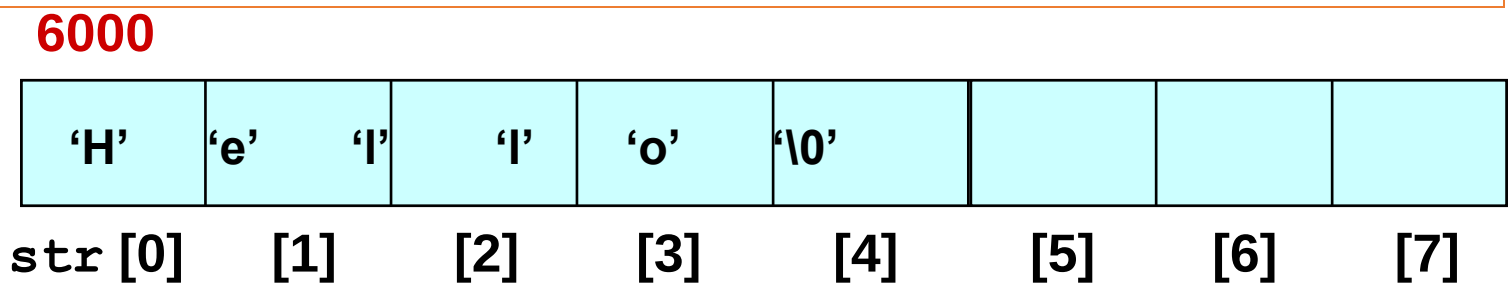
C++ Data Types


simple
structured
integral
enum
floating
array
struct
union
class
char
short
int
long
bool
float
double
long double
address
pointer
reference

Recall that . . .

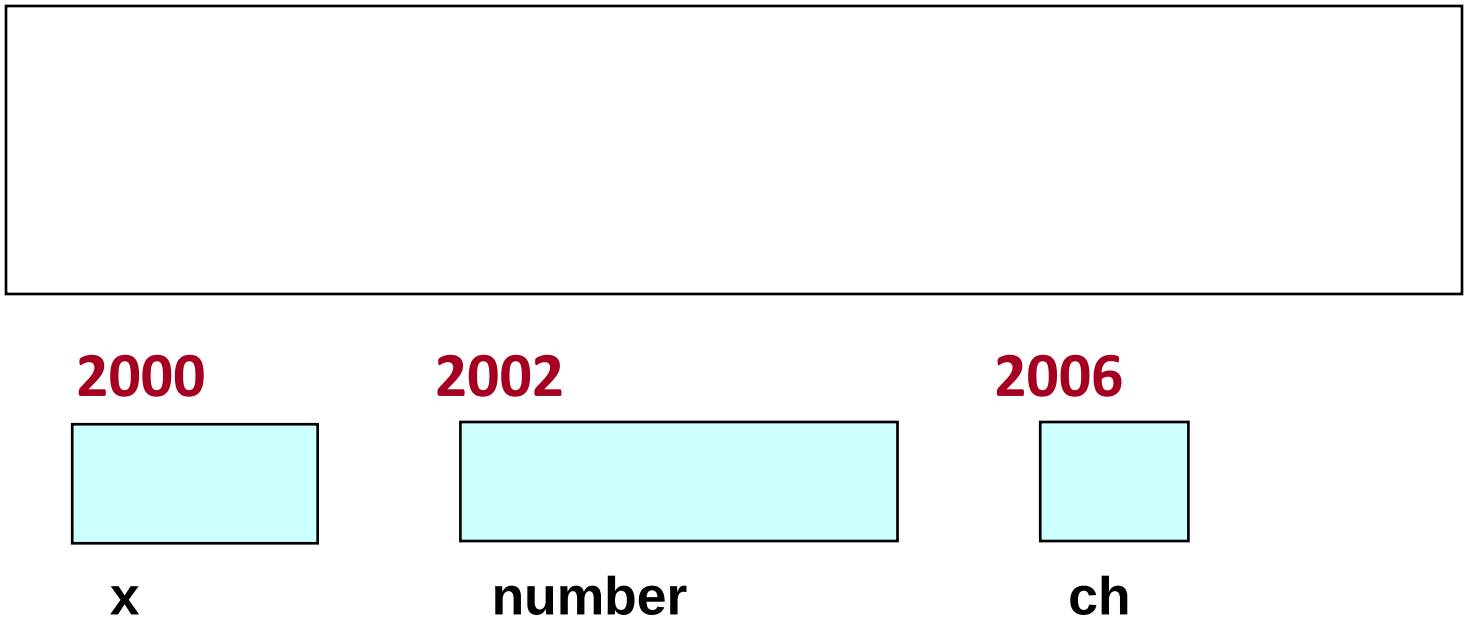
```
char  str [ 8 ];
```

- **`str`** is the base address of the array.
- We say **`str`** is a pointer because its value is an address.
- It is a pointer constant because the value of **`str`** itself cannot be changed by assignment. It "points" to the memory location of a `char`.



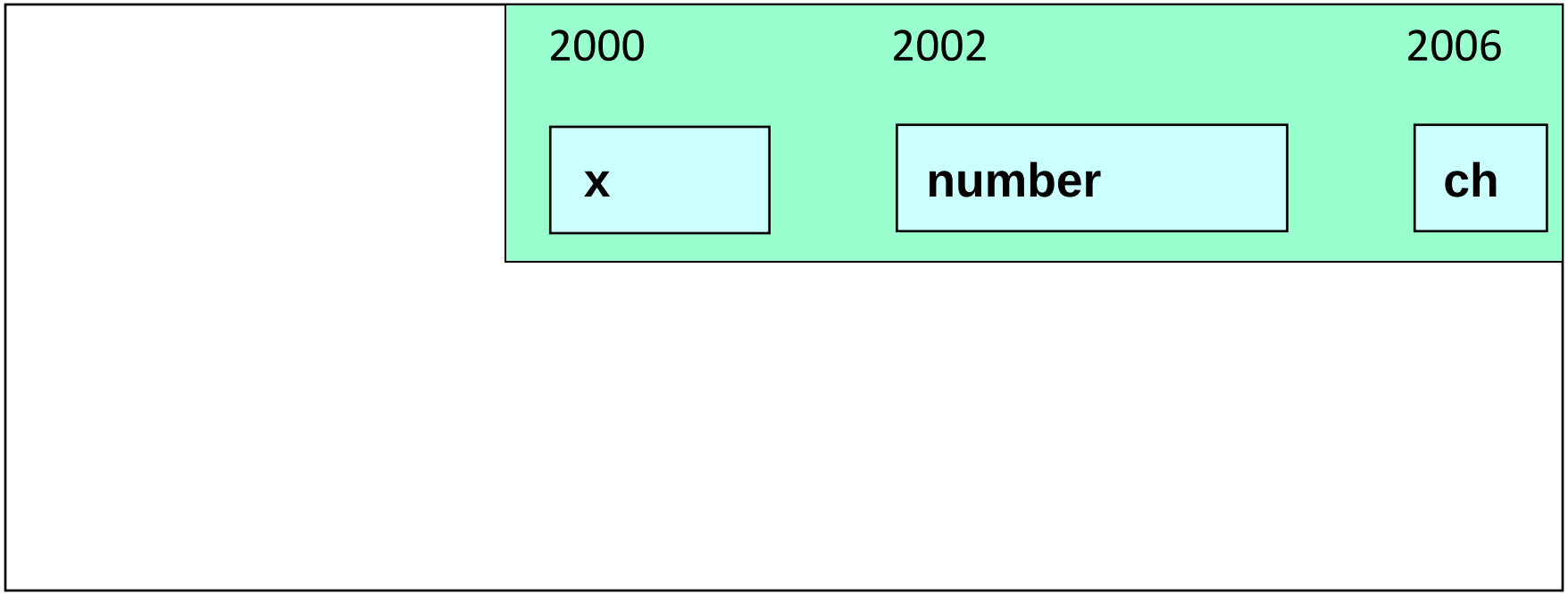
Addresses in Memory

- When a variable is declared, enough memory to hold a value of that type is allocated for it at an unused memory location. This is the address of the variable



Obtaining Memory Addresses

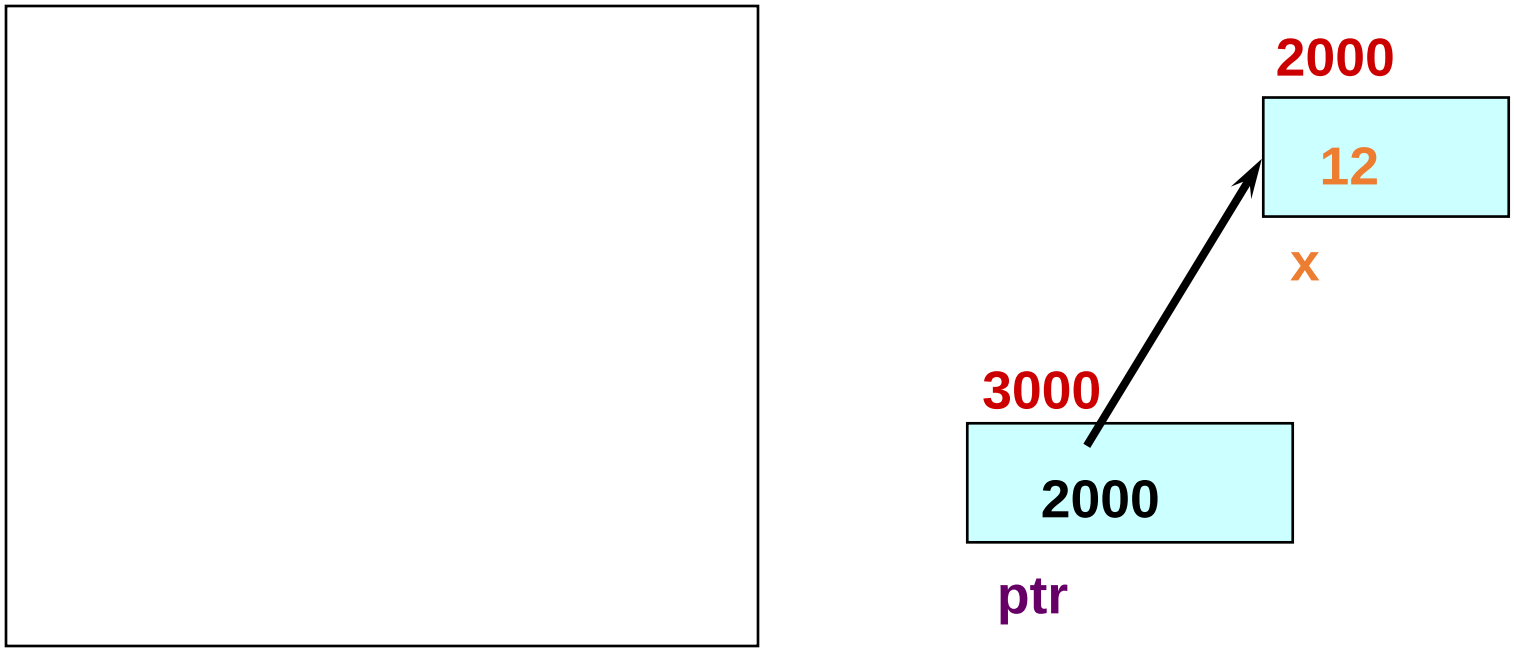
- The address of a *non-array variable* can be obtained by using the address-of operator &



What is a pointer variable?

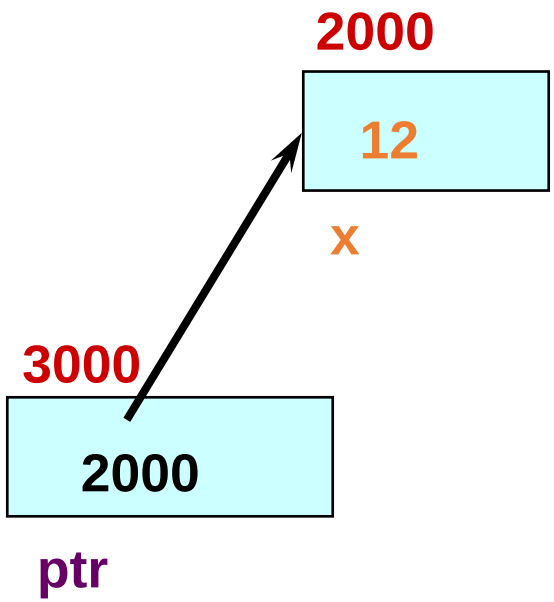
- A pointer variable is a variable whose value is the address of a location in memory.
- To declare a pointer variable, you must specify the type of value that the pointer will point to, for example,

Using a Pointer Variable



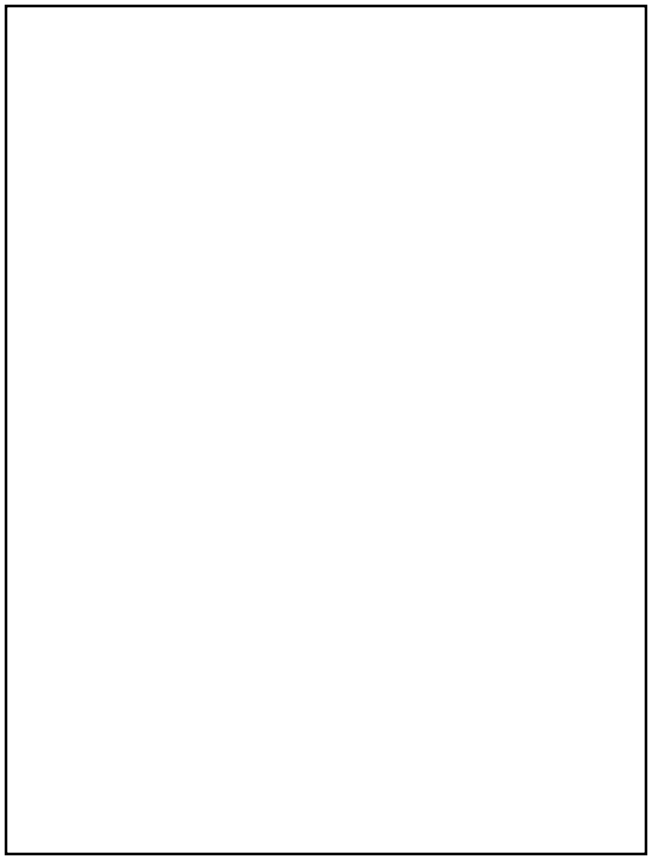
NOTE: **Because ptr holds the address of x, we say that ptr "points to" x**

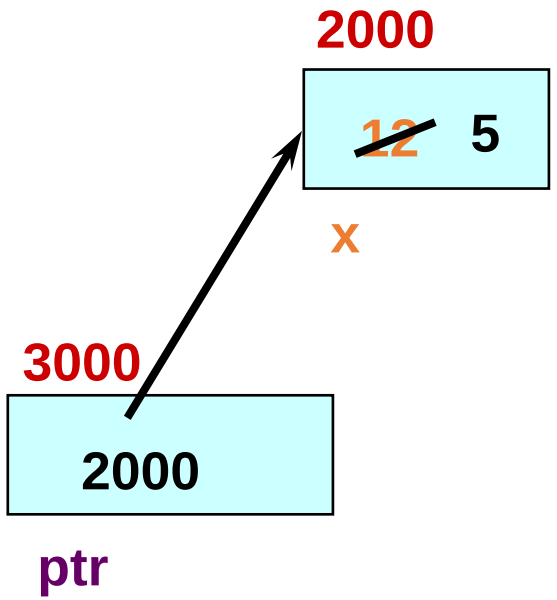
*: dereference operator



NOTE: **The value pointed to by ptr is denoted by *ptr**

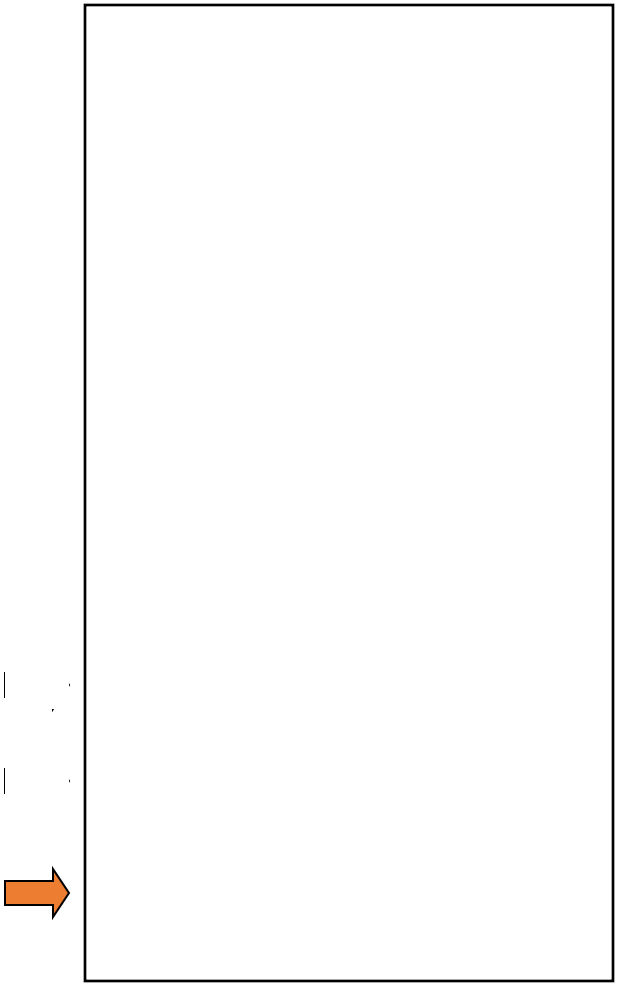
Using the Dereference Operator

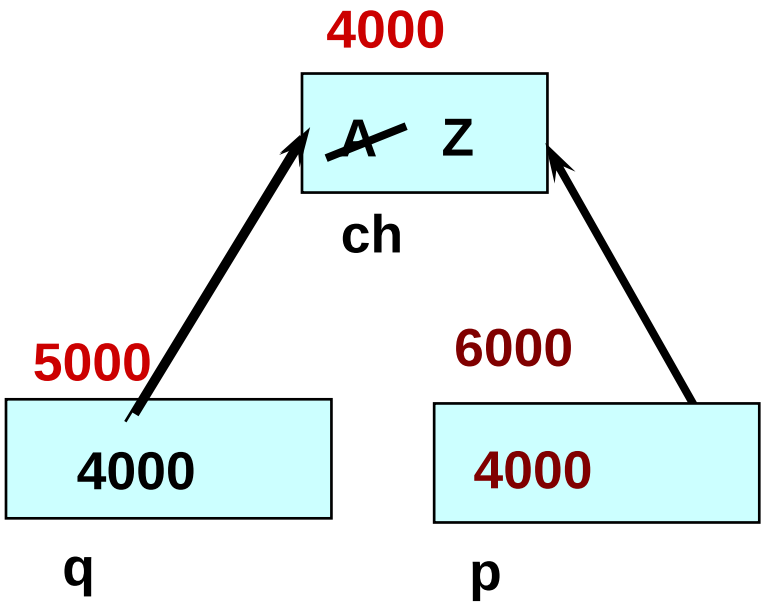





```
// changes the value at the
   address ptr points to 5
```

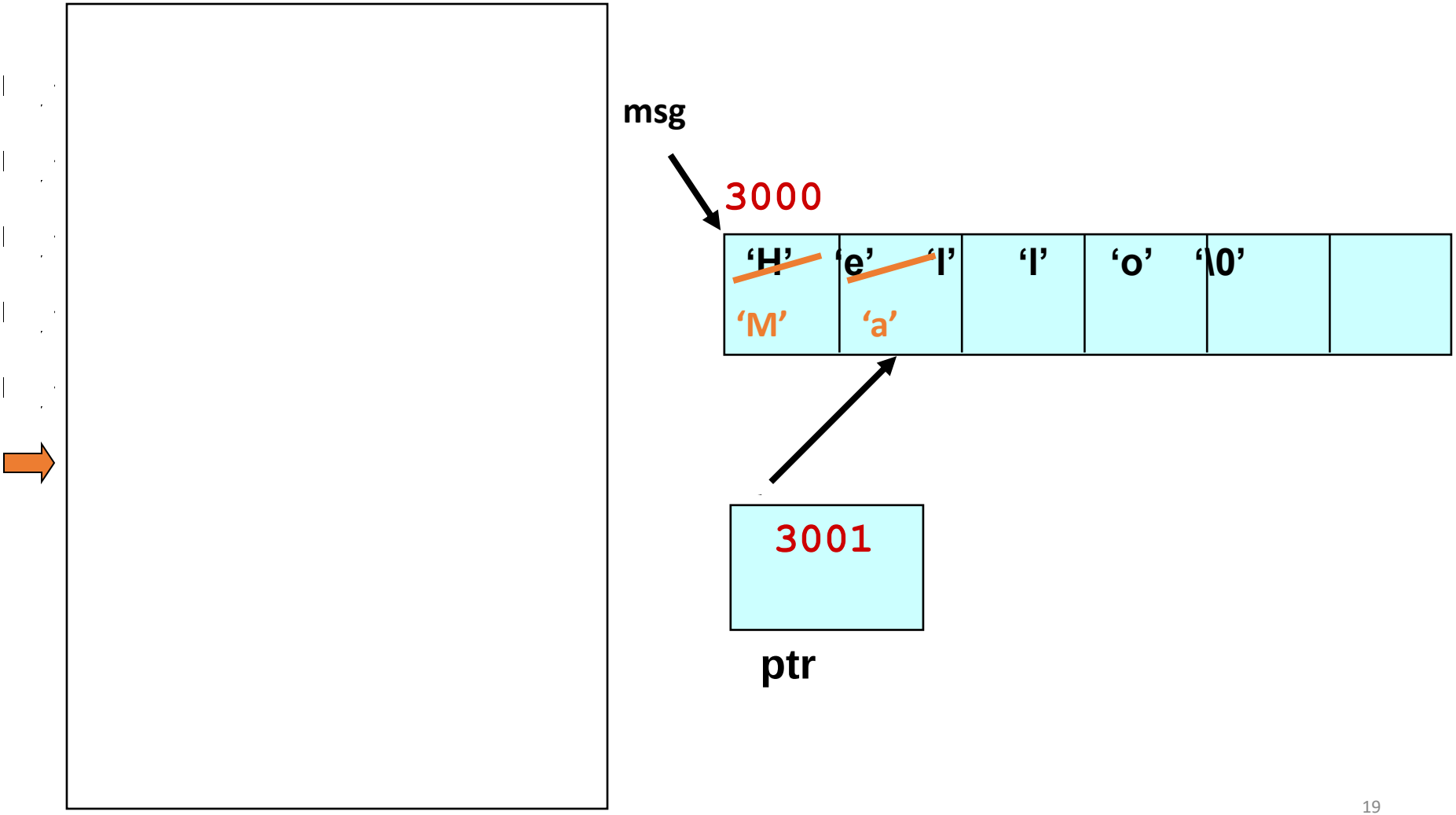
Self –Test on Pointers






```
// the rhs has value 4000

// now p and q both point to ch
```

Using a Pointer to Access the Elements of a String



Reference Variables

Reference variable = ***alias for another variable***

- Contains the address of a variable (like a pointer)
- No need to perform any dereferencing (unlike a pointer)
- Must be initialized when it is declared

```
int x = 5;
int &z = x;            // z is another name for x
int &y ;         //Error: reference must be initialized
cout << x << endl;   -> prints 5
cout << z << endl;   -> prints 5

z = 9;               // same as x = 9;

cout << x << endl;   -> prints 9
cout << z << endl;   -> prints 9
```

Why Reference Variables

- Are primarily used as function parameters
- Advantages of using references:
  - you don't have to pass the address of a variable
  - you don't have to dereference the variable inside the called function

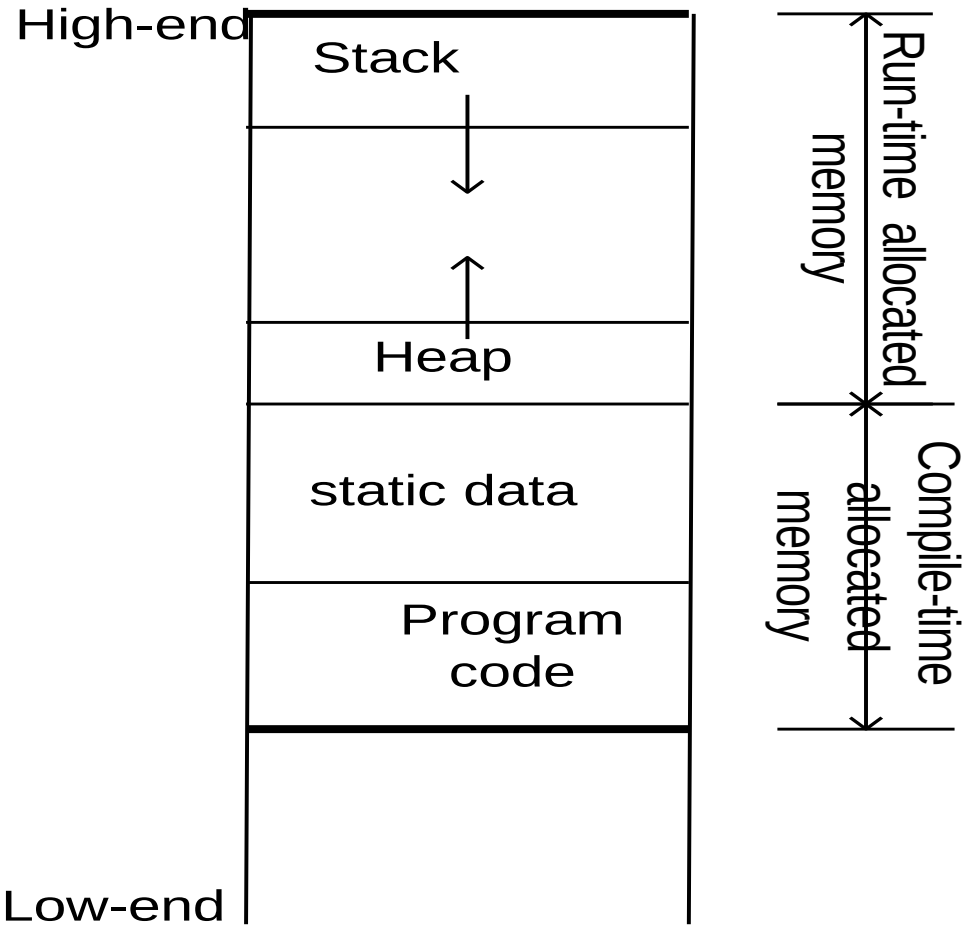
Reference Variables Example

```
#include <iostream.h>

// Function prototypes
  (required in C++)

void p_swap(int *, int *);
void r_swap(int&, int&);

int main (void){
 int v = 5, x = 10;
 cout << v << x << endl;
 p_swap(&v,&x);
 cout << v << x << endl;
 r_swap(v,x);
 cout << v << x << endl;
 return 0;
}
```

```
void p_swap(int *a, int *b)
{
  int temp;
  temp = *a;      (2)
  *a = *b;        (3)
  *b = temp;

}
```

```
void r_swap(int &a, int &b)
{
 int temp;
 temp = a;        (2)
 a = b;           (3)
 b = temp;
}
```

Dynamic Memory Allocation Diagram



Dynamic Memory Allocation

- *In C*, functions such as malloc() are used to dynamically allocate memory from the **Heap**.
- *In C++,* this is accomplished using the **new** and **delete** operators
- **new** is used to allocate memory during execution time
  - returns a pointer to the address where the object is to be stored
  - always returns a pointer to the type that follows the **new**

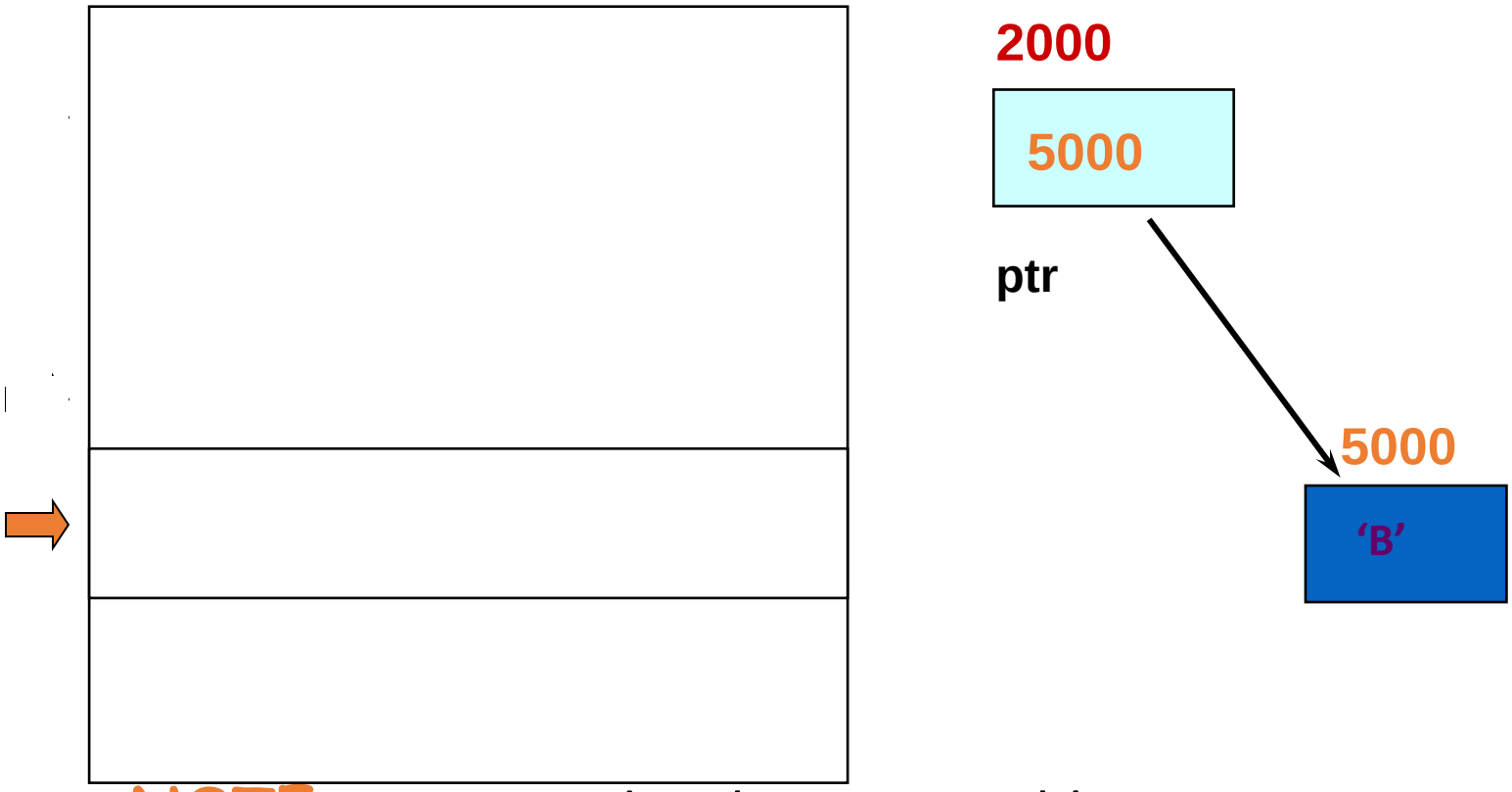
Operator `new` Syntax

```
new   DataType
```

```
new   DataType  [IntExpression]
```

- If memory is available, in an area called the heap (or free store) new allocates the requested object or array, and returns a pointer to (address of) the memory allocated.
- Otherwise, program terminates with error message.
- The dynamically allocated object exists until the delete operator destroys it.

Operator **new**



NOTE: Dynamic data has no variable name

The NULL Pointer

- There is a pointer constant called the “null pointer” denoted by NULL
- But NULL is not memory address 0.

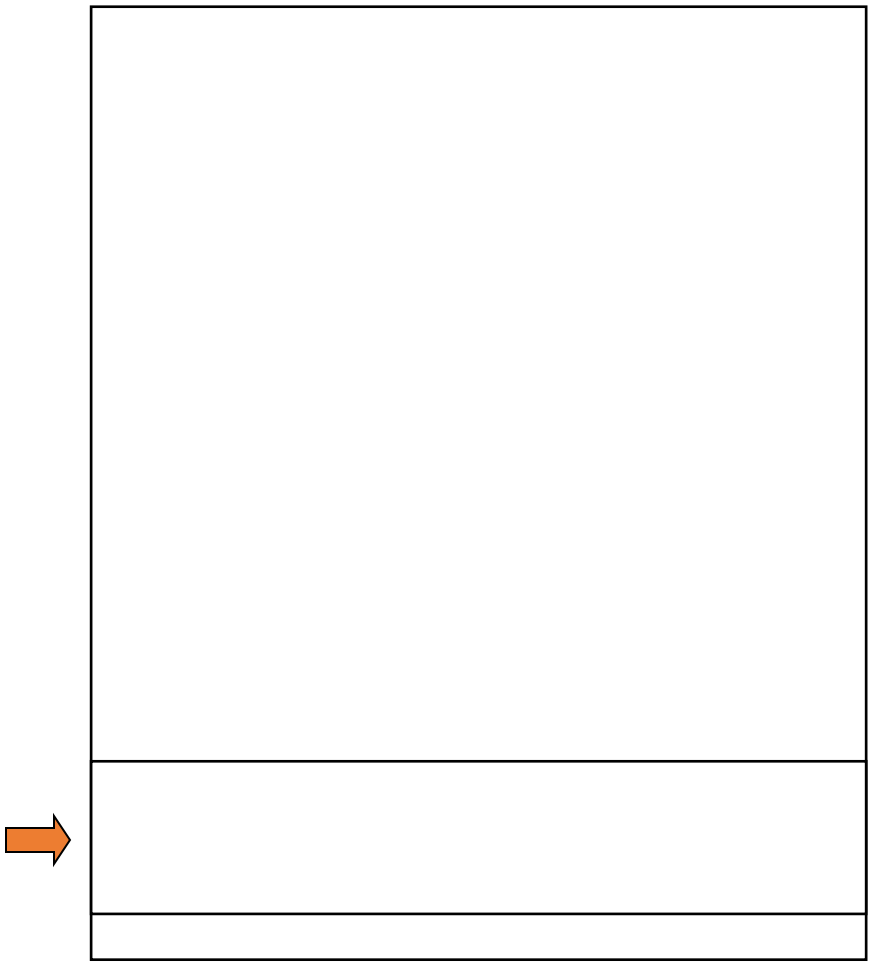
- NOTE: It is an error to dereference a pointer whose value is NULL. Such an error may cause your program to crash, or behave erratically. It is the programmer’s job to check for this.

```
while (ptr != NULL) {
    . . .        // ok to use *ptr here
}
```

Operator `delete` Syntax

- The object or array currently pointed to by Pointer is deallocated, and the value of Pointer is undefined. The memory is returned to the free store.
- Good idea to set the pointer to the released memory to NULL
- Square brackets are used with delete to deallocate a dynamically allocated array.

Operator delete



2000

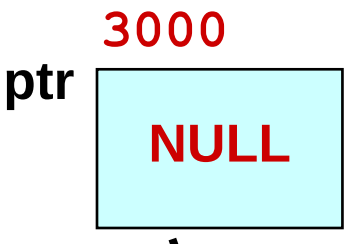
ptr

NOTE:

delete deallocates the memory pointed to by ptr

Example

```
char  *ptr ;

ptr  =  new  char[ 5 ];

strcpy( ptr, “Bye” );

ptr[ 0 ] = ‘u’;


delete [] ptr;

ptr = NULL;
```



// deallocates the array pointed to by ptr
// ptr itself is not deallocated
// the value of ptr becomes undefined

Pointers and Constants

```
char* p;
p = new char[20];

char c[] = “Hello”;
const char* pc = c; //pointer to a constant
pc[2] = ‘a’; // error
pc = p;

char  *const cp = c; //constant pointer
cp[2] = ‘a’;
cp = p; // error

const char *const cpc = c; //constant pointer to a const
cpc[2] = ‘a’; //error
cpc = p; //error
```

Thank You