**Topics:**

1. Pointers and Dynamic memory
2. Pointers and Arrays as parameters
3. Classes and Dynamic Array

**Variable Names and address**

* direct addressing.
* A pointer is a memory address of a variable

also called indirect addressing

* Declaring a point Variable

syntax

<data Type>\* <pointer name>

or

<data Type> \*<pointer name>

ex:

int\* IntPointer

int \*IntPointer

char \*CharPointer

* Declaring more than one pointers

int \*P ,\*Q;

or

typedef int\* ptrType;

ptrType P, Q;

**Assigning Values to a Pointer Variable**

**1. Static Assignment**

int\* intPointer

int num = 5;

intPointer = &num

**Dereferencing operator**

\*intPointer = 6;

cout << \*intPointer

**Assignment operator with pointer**

Example 1:

int \*p, \*q;

int x=5;

p=&x;

\*p=6

**q=p;**

cout << x << \*p **<< \***q

Example 2:

int \*p, \*q;

int x=5, y = 10;

p=&x;

q=&y;

**\*q = \*p**

**2. Dynamically Allocated Variable**

Dynamic variables are created during execution

using the operator *new*

**Dynamic assignment**

Syntax:

<pointer variable> = new <type>

**The Heap**

1. A special area in memory is reserved for dynamic variables and dynamic arrays.

1. If your program consumes all of the heap memory, an additional calls to *new* will fail.

1. **NULL** is a special constant that is used to give a value to a pointer variable that would not otherwise have a value
2. Null can be assigned to any pointer of any type

1. The definition of the identifier NULL is in <csdtdef>

Examples:

1. int \*intPointer;

intPointer= new int;

\*intPointer = 10;

1. char \*char\_Pointer = new char;

\*char\_Pointer = ‘A’;

**3.**

int main()

{

int \*p1,\*p2;

p1 = new int;

\*p1 = 42

//\*p2 = 100

p2= p1;

\*p2 = 53;

p1 = new int;

\*p1 = 88;

return 0;

}

**The *delete* operator**

The delete operator eliminates a dynamic variable and returns the memory that the dynamic variable occupied to the heap.

1. The memory can then be reused to create new dynamic variable.
2. Syntax

delete <pointer variable>

#include <iostream>

using namespace std;

int main()

{

int \*p1,\*p2;

p1 = new int;

\*p1 = 42;

p2= p1;

cout <<"1. "<< "p1 = "<<p1 << " \*p1= "<<\*p1<<

" " << "p2 = "<<p2 << " \*p2= "<<\*p2<< endl;

/\*

output:

1. p1 = 00424C38 \*p1= 42 p2 = 00424C38 \*p2= 42

\*/

delete p1; //the data at the memory location is deleted

// delete p2; will give an error since p2 is already deleted

cout <<"2. "<< "p1 = "<<p1 << " \*p1= "<<\*p1<<

" " << "p2 = "<<p2 << " \*p2= "<<\*p2<< endl;

/\*output:

2. p1 = 00424C38 \*p1= -17891602 p2 = 00424C38 \*p2= -17891602

\*/

\*p2 = 53; //store data in the memory location ( were the data

//before was deleted

cout <<"3. "<< "p1 = "<<p1 << " \*p1= "<<\*p1<<

" " << "p2 = "<<p2 << " \*p2= "<<\*p2<< endl;

/\*output:

3. p1 = 00424C38 \*p1= 53 p2 = 00424C38 \*p2= 53

\*/

p1 = new int;

\*p1 = 88;

cout <<"4. "<< "p1 = "<<p1 << " \*p1= "<<\*p1<<

" " << "p2 = "<<p2 << " \*p2= "<<\*p2<< endl;

/\*output:

4. p1 = 00424C38 \*p1= 88 p2 = 00424C38 \*p2= 88

\*/

p2 = new int;

\*p2 = 100;

cout <<"5. "<< "p1 = "<<p1 << " \*p1= "<<\*p1<<

" " << "p2 = "<<p2 << " \*p2= "<<\*p2<< endl;

/\*output:

5. p1 = 00424C38 \*p1= 88 p2 = 00424CC8 \*p2= 100

\*/

system("pause");

return 0;

}

/\* output:

1. p1 = 00424C38 \*p1= 42 p2 = 00424C38 \*p2= 42

2. p1 = 00424C38 \*p1= -17891602 p2 = 00424C38 \*p2= -17891602

3. p1 = 00424C38 \*p1= 53 p2 = 00424C38 \*p2= 53

4. p1 = 00424C38 \*p1= 88 p2 = 00424C38 \*p2= 88

5. p1 = 00424C38 \*p1= 88 p2 = 00424CC8 \*p2= 100

Press any key to continue . . .

\*/

**Arrays**

//Program to demonstrate that an array variable is a kind of pointer variable.

#include <iostream>

using namespace std;

**typedef int\* IntPtr;**

int main( )

{

IntPtr p;

int a[10];

int index;

for (index = 0; index < 10; index++)

a[index] = index;

**p = a;**

for (index = 0; index < 10; index++)

cout << p[index] << " ";

cout << endl;

for (index = 0; index < 10; index++)

p[index] = p[index] + 1;

for (index = 0; index < 10; index++)

cout << a[index] << " ";

cout << endl;

return 0;

}

**Dynamic arrays**

1. When a *new* operator allocates an entire array, it returns a pointer to the first element of the array
2. Example

int \*arr\_ptr;

cin >>size;

arr\_ptr = new int[size];

arr\_ptr[1] = 10;

1. use *delete* operator to release a dynamic allocated array

delete[] arr\_ptr;

Example:

//Sorts a list of numbers entered at the keyboard.

#include <iostream>

#include <cstdlib>

#include <cstddef>

using namespace std;

**typedef int\* IntArrayPtr;**

**void fill\_array(int a[], int size);**

//Precondition: size is the size of the array a.

//Postcondition: a[0] through a[size-1] have been

//filled with values read from the keyboard.

**void sort(int a[], int size);**

//Precondition: size is the size of the array a.

//The array elements a[0] through a[size-1] have //values.

//Postcondition: The values of a[0] through a[size-1] //have been rearrangedso that

// a[0] <= a[1] <= ... <= a[size-1].

**void swap\_values(int& v1, int& v2);**

int index\_of\_smallest(const int a[], int start\_index, int number\_used);

**int main( )**

{

cout << "This program sorts numbers from lowest"

<<" to highest.\n";

int array\_size;

cout << "How many numbers will be sorted? ";

cin >> array\_size;

**IntArrayPtr a;**

**a = new int[array\_size];**

fill\_array(a, array\_size);

sort(a, array\_size);

cout << "In sorted order the numbers are:\n";

for (int index = 0; index < array\_size; index++)

cout << a[index] << " ";

cout << endl;

**delete [] a;**

**//delete a[] is incorrect**

return 0;

}

//Uses the library iostream:

**void fill\_array(int a[], int size)**

**// or void fill\_array(int\* a, int size)**

{

cout << "Enter " << size << " integers.\n";

for (int index = 0; index < size; index++)

cin >> a[index];

}

**void sort(int a[], int size)**

{ // use here a sorting algorithm}

* A function can return a pointer , but not an array

int[] func1(); // not ok

int\* func2(); // ok

**Multidimensional array ( is an aray of arrays)**

Example of an array of 3X4

typedef int \* ArrayPtr;

ArrayPtr \*m = new ArrayPtr[3];

Now we can do

m[0] = new int[4];

m[1] = new int[4];

m[2] = new int[4];

or

for ( int i = 0; i<3, i++)

m[i] = new int[4];

**to make m dynamic array of 3x4**

cout << “enter array size”;

cin >>size1 >> size2 // for (size1xsize2 array-– size1 rows, size2 columns)

ArrayPtr \*m = new ArrayPtr[size1];

for ( int i = 0; i<=size2, i++)

m[i] = new int[size2];

:

at the end need to delete

for ( int i = 0; i<=size2, i++)

delete [] m[i];

delete [] m

**Passing Pointers**

1. **Pointer as a value parameter**

**I.**

int\* int-ptr = new int;

\*int\_ptr = 10;

function\_A(int\_ptr); //a call to the function

void function\_A(int\* i\_ptr) // function definition

{

\*i\_ptr = 20;

}

**II. void function\_B(const int\* i\_ptr);**

1. **Pointers as reference Parameters**

int\* int\_ptr;

the call:

function\_C(int\_ptr)

void function\_C(int\*& i\_ptr)

{

i\_ptr = new int;

}

1. **Array Parameters**

int List[4]={1,2,3,4};

void function\_D(const int arr[],int size)

{

for (int i = 0; i<size; i++)

cout<< arr[i];

}

the call:

function\_D(List,4);

**Functions that returns an array**

int[] somefunc () // illegal

int\* somefunc () // ok

**call the function**

int x func (x) void func(int y) x y

5

5

5

int x func (x) void func(int& y) x , y

5

A0

P

A0

A0

q

int \*p func (p) void func(int \*q)

\*p = 5 { \*q = 6}

int \*p func (p) void func(int \*&q)

P

\*p = 5 { \*q = new int} q

5

A0

A0

int \*p int\* z= func (p) int\* func(int \*q)

\*p = 5 { q =new int

P, q

\*q =10

A1

A1

Z

A0

A0

5

return q

}

10

**Dynamic class that has dynamic array**

//This is the HEADER FILE strvar.h.

//This is the INTERFACE for the class StringVar whose values

//are strings. An object is declared as follows.

//Note that you use (max\_size), not [max\_size]

// StringVar the\_object(max\_size);

//where max\_size is the longest string length allowed. The argument max\_size can be a variable.

#ifndef STRVAR\_H

#define STRVAR\_H

#include <iostream>

using namespace std;

namespace savitchstrvar

{

class StringVar

{

public:

StringVar(int size);

//Initializes the object so it can accept string values up to size

// in length.Sets the value of the object equal to the empty

//string.

StringVar( );

//Initializes the object so it can accept string values of length

// 100 or less.

//Sets the value of the object equal to the empty string.

StringVar(const char a[]);

//Precondition: The array a contains characters terminated

// with ’\0’.Initializes the object so its value is the string

// stored in a and so that it can later be set to string values up

//to strlen(a) in length

StringVar(const StringVar& string\_object);

//Copy constructor.

~StringVar( );

//Returns all the dynamic memory used by the object to the

//heap.

int length( ) const;

//Returns the length of the current string value.

void input\_line(istream& ins);

//Precondition: If ins is a file input stream, then ins has

//already been connected to a file.

//Action: The next text in the input stream ins, up to ’\n’, is

//copied to the calling object.

// If there is not sufficient room, then only as much as will fit

//is copied.

friend ostream& operator <<(ostream& outs, const

StringVar& the\_string);

//Overloads the << operator so it can be used to output values

//of type StringVar

//Precondition: If outs is a file output stream, then outs

//has already been connected to a file.

private:

char \*value; //pointer to the dynamic array that holds the

//string value.

int max\_length; //declared max length of any string value.

};

}//savitchstrvar

#endif //STRVAR\_H

//This is the IMPLEMENTATION FILE: strvar.cxx

//This is the IMPLEMENTATION of the class StringVar.

//The interface for the class StringVar is in the header file strvar.h.

#include <iostream>

#include <cstdlib>

#include <cstddef>

#include <cstring>

#include "strvar.h"

namespace savitchstrvar

{

//Uses cstddef and cstdlib:

StringVar::StringVar(int size)

{

max\_length = size;

value = new char[max\_length + 1];//+1 is for '\0'.

value[0] = '\0';

}

//Uses cstddef and cstdlib:

StringVar::StringVar( )

{

max\_length = 100;

value = new char[max\_length + 1];//+1 is for '\0'.

value[0] = '\0';

}

//Uses cstring, cstddef, and cstdlib:

StringVar::StringVar(const char a[])

{

max\_length = strlen(a);

value = new char[max\_length + 1];//+1 is for '\0'.

strcpy(value, a);

}

**//Uses cstring, cstddef, and cstdlib:**

**StringVar::StringVar(const StringVar& string\_object)**

**{**

**max\_length = string\_object.length( );**

**value = new char[max\_length + 1];//+1 is for '\0'.**

**strcpy(value, string\_object.value);**

**}**

**StringVar::~StringVar( )**

**{**

**delete [] value;**

**}**

//Uses cstring:

int StringVar::length( ) const

{

return strlen(value);

}

//Uses iostream:

void StringVar::input\_line(istream& ins)

{

ins.getline(value, max\_length + 1);

}

//Uses iostream:

ostream& operator <<(ostream& outs,

const StringVar& the\_string)

{

outs << the\_string.value;

return outs;

}

}//savitchstrvar

/

/Program to demonstrate use of the class StringVar.

#include <iostream>

#include "strvar.h"

using namespace std;

using namespace savitchstrvar;

void conversation(int max\_name\_size);

//Carries on a conversation with the user.

**int main( )**

{

**conversation(30);**

cout << "End of demonstration.\n";

return 0;

}

// This is only a demonstration function:

void conversation(int max\_name\_size)

{

**StringVar your\_name(max\_name\_size), our\_name("Borg");**

cout << "What is your name?\n";

your\_name.input\_line(cin);

cout << "We are " << our\_name << endl;

cout << "We will meet again " << your\_name << endl;

}

**Destructor**

1. The destructor is a class member function that releases dynamically allocated memory once the object is inaccessible
2. the destructor is automatically activated.
3. the destructor has the name of the class precede by the ~ symbol

the destructor does not return a value.

**copy constructor**

* is a constructor that has one parameter that is the same type as the class
* that parameter must be a call by reference and normally it precede with the modifier const
* the copy constructor is called automatically when :
* a function returns a value of the class type,
* and whenever the parameter of a function is call-by-value of the class type.
* An object is initialized to another object
* Any class that uses pointer and the new operator should have the copy constructor

**Overloading assignment operator**

void StringVar::operator=(const StringVar & rhs)

{

int new\_length = strlen(rhs.value);

if (new\_length > max\_length) //not enough room in lhs

{

delete [ ] value; //deallocate lhs space

max\_length = new\_length;

value = new char[max\_length + 1];//allocate space

}

for(int i = 0; i < new\_length; i++)//have space now,

value[i] = rhs.value[i]; //copy data.

value[new\_length] = '\0';

}