DR. (MRS) T-S.M.A. ADJAIDOO

07
Pointers

Today's Lesson

Learning Outcomes

1

- Learners must have a general understanding of what pointers are

2

- Learners must be able to use pointers for various purposes in coding

Pointers

Pointers

Some common uses:

- Accessing array elements
- Passing arguments to a function when the function needs to modify the original argument
- Passing arrays and strings to functions
- Obtaining memory from the system C
- Creating data structures such as linked lists

Pointers

- Computer memory can be imagined as a very large array of bytes.
- A variable declaration associates three fundamental attributes to the variable:
  - its name, its type and its memory address.
- The address of a variable can be obtained using the reference operator &, also called the address operator.

Pointers

Code

```
#include <iostream>
using namespace std;
int main()
{
    int n=20;
    cout<<"n="<<n<<endl;
    cout<<"&n="<<&n<<endl;
}
```

Output

```
n=20
&n=0x7ffeefbff81c
```

References

- A reference is an alias or synonym for another variable.
- It is declared by the syntax

  type& ref-name = var-name;

  where type is the variable's type, ref-name is the name of the reference and var-name is the name of the variable

References

Code

```
#include <iostream>
using namespace std;
int main()
{
    int n=20;
    int& rn = n; //rn is a synonym for n

    cout<<"n="<<n<<", rn ="<<rn<<endl;
    --n;
    cout<<"n="<<n<<", rn ="<<rn<<endl;
    rn*=2;
    cout<<"n="<<n<<", rn ="<<rn<<endl;
}
```

Output

```
n=20, rn =20
n=19, rn =19
n=38, rn =38
```

Pointer Variables

A variable that holds an address value is called a *pointer variable, or simply a pointer.*

```
#include <iostream>
using namespace std;
int main()
{
    int n=20;
    cout<<"n="<<n<<"\n&n="<<&n<<endl;
    int* pn =&n;

    cout<<"pn="<<pn<<endl;
    cout<<"&pn="<<&pn<<endl;
}
```

```
n=20
&n=0x7ffeefbff81c
pn=0x7ffeefbff81c
&pn=0x7ffeefbff810
```

The dereference operator

- When an asterisk is used in front of a variable name, it is called the dereference operator (or sometimes the indirection operator).
- It means the value of the variable pointed to by.

The dereference operator

```cpp
#include <iostream>
using namespace std;
int main()
{
    int var1 = 11;
    int var2 = 22;

    int* ptr;
    ptr = &var1;
    cout << *ptr << endl;

    ptr = &var2;
    cout << *ptr << endl;
    return 0;
}
```

```cpp
#include <iostream>
using namespace std;
int main()
{
    int var1, var2;
    int* ptr;
    ptr = &var1;
    *ptr = 37;
    var2 = *ptr;
    cout << var2 << endl;
    return 0;
}
```

Pointer to void

- Ordinarily, the address that you put in a pointer must be the same type as the pointer.
- You can't assign the address of a float variable to a pointer to int.
- However, there is an exception to this. There is a sort of general-purpose pointer that can point to any data type.
- This is called a pointer to void, and is defined like this:

```
void* ptr; //ptr can point to any data type
```

- Such pointers have certain specialized uses, such as passing pointers to functions that operate independently of the data type pointed to.

Pointer to void

Look at this example

```
// pointers to type void
#include <iostream>
using namespace std;

int main()
{
    int intvar; //integer variable
    float flovar; //float variable

    int* ptrint; //define pointer to int
    float* ptrflo; //define pointer to float
    void* ptrvoid; //define pointer to void
        ptrint = &intvar; //ok, int* to int*
        // ptrint = &flovar; //error, float* to int*
        // ptrflo = &intvar; //error, int* to float*
        ptrflo = &flovar; //ok, float* to float*
        ptrvoid = &intvar; //ok, int* to void*
        ptrvoid = &flovar; //ok, float* to void*
    return 0;
}
```

Pointer to void

- You can assign the address of intvar to ptrint because they are both type int*, but you can’t assign the address of flovar to ptrint because the first is type float* and the second is type int*.
- However, ptrvoid can be given any pointer value, such as int*, because it is a pointer to void.

Null pointer

- A null pointer is a regular pointer of any pointer type which has a special value that indicates that it is not pointing to any valid reference or memory address.
- This value is the result of type-casting the integer value zero to any pointer type.
  - *int* * p;
  - p = 0; *// p has a null pointer value*
- Do not confuse null pointers with void pointers.
- A null pointer is a value that any pointer may take to represent that it is pointing to "nowhere", while a void pointer is a special type of pointer that can point to somewhere without a specific type.
- One refers to the value stored in the pointer itself and the other to the type of data it points to.

Pointer to Pointers

Output:

```
n=44
&n=0x7ffeefbff81c
pn=0x7ffeefbff81c
&pn=0x7ffeefbff810
*pn=44
ppn=0x7ffeefbff810
&ppn=0x7ffeefbff808
*ppn=0x7ffeefbff81c
**ppn=44
```

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

int main()
{
    int n = 44;
    cout<<"n="<<n<<endl;
    cout<<"&n="<<&n<<endl;

    int* pn = &n;
    cout<<"pn="<<pn<<endl;
    cout<<"&pn="<<&pn<<endl;
    cout<<"*pn="<<*pn<<endl;

    int** ppn = &pn;
    cout<<"ppn="<<ppn<<endl;
    cout<<"&ppn="<<&ppn<<endl;
    cout<<"*ppn="<<*ppn<<endl;
    cout<<"**ppn="<<**ppn<<endl;
}
```

Pointers to functions

- C++ allows operations with pointers to functions.
- The typical use of this is for passing a function as an argument to another function, since these cannot be passed dereferenced.
- In order to declare a pointer to a function we have to declare it like the prototype of the function except that the name of the function is enclosed between parentheses () and an asterisk (*) is inserted before the name

Pointers to functions

In the example, minus is a pointer to a function that has two parameters of type int.

It is immediately assigned to point to the function subtraction, all in a single line:

int (* minus)(*int*,*int*) = subtraction;

```
// pointer to functions
#include <iostream>
using namespace std;
int addition (int a, int b) {
    return (a+b);
}
int subtraction (int a, int b) {
    return (a-b);
}

int operation (int x, int y, int (*functocall)(int,int)) {
    int g;
    g = (*functocall)(x,y);
    return (g);
}
int main ()
{
    int m,n;
    int (*minus)(int,int) = subtraction;
    m = operation (7, 5, addition);
    n = operation (20, m, minus);
    cout <<n;
    return 0;
}
```

The new Operator

- C++ provides a different approach to obtaining blocks of memory: the new operator.
- This versatile operator obtains memory from the operating system and returns a pointer to its starting point.

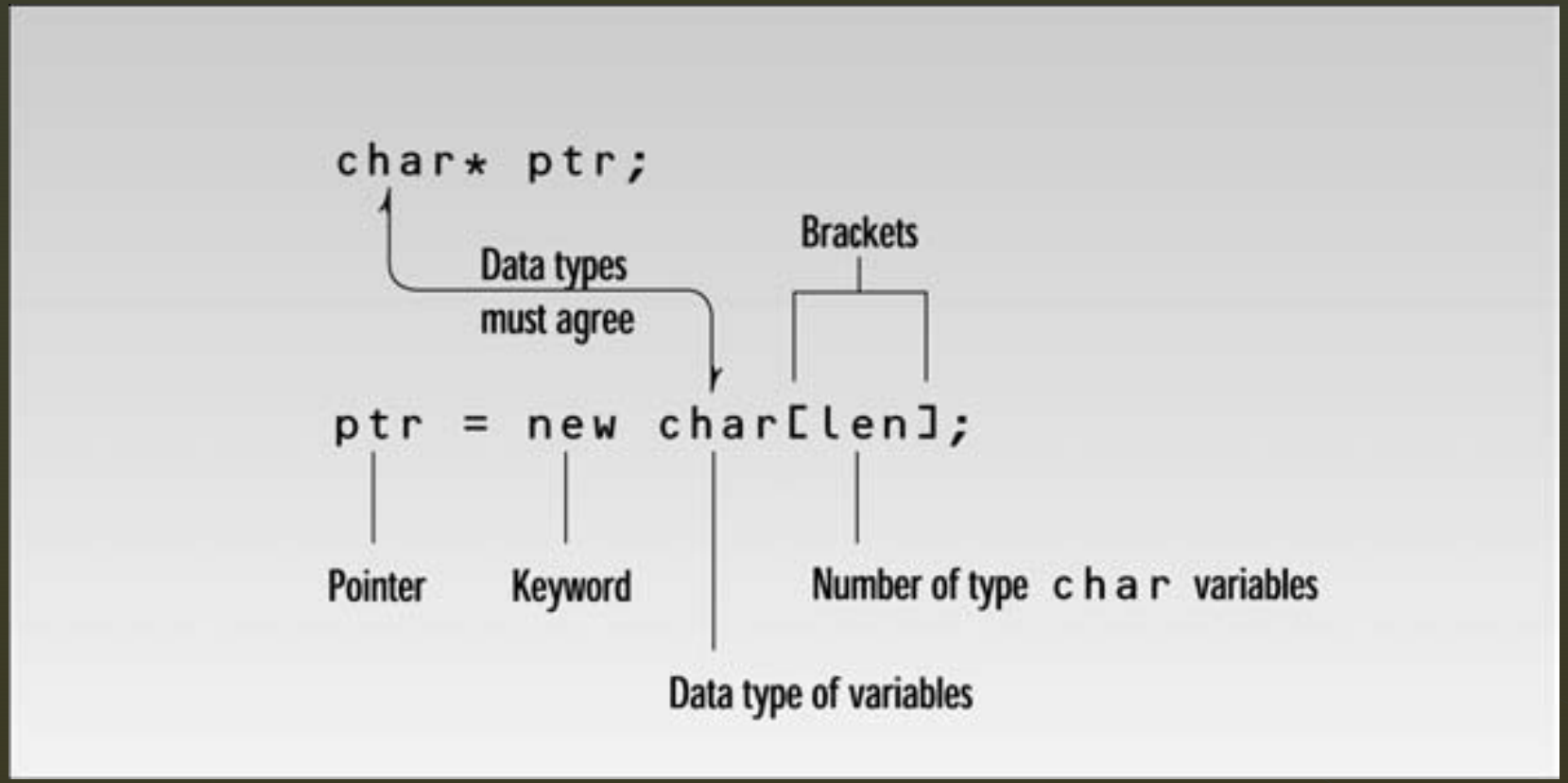
The new Operator

The expression:

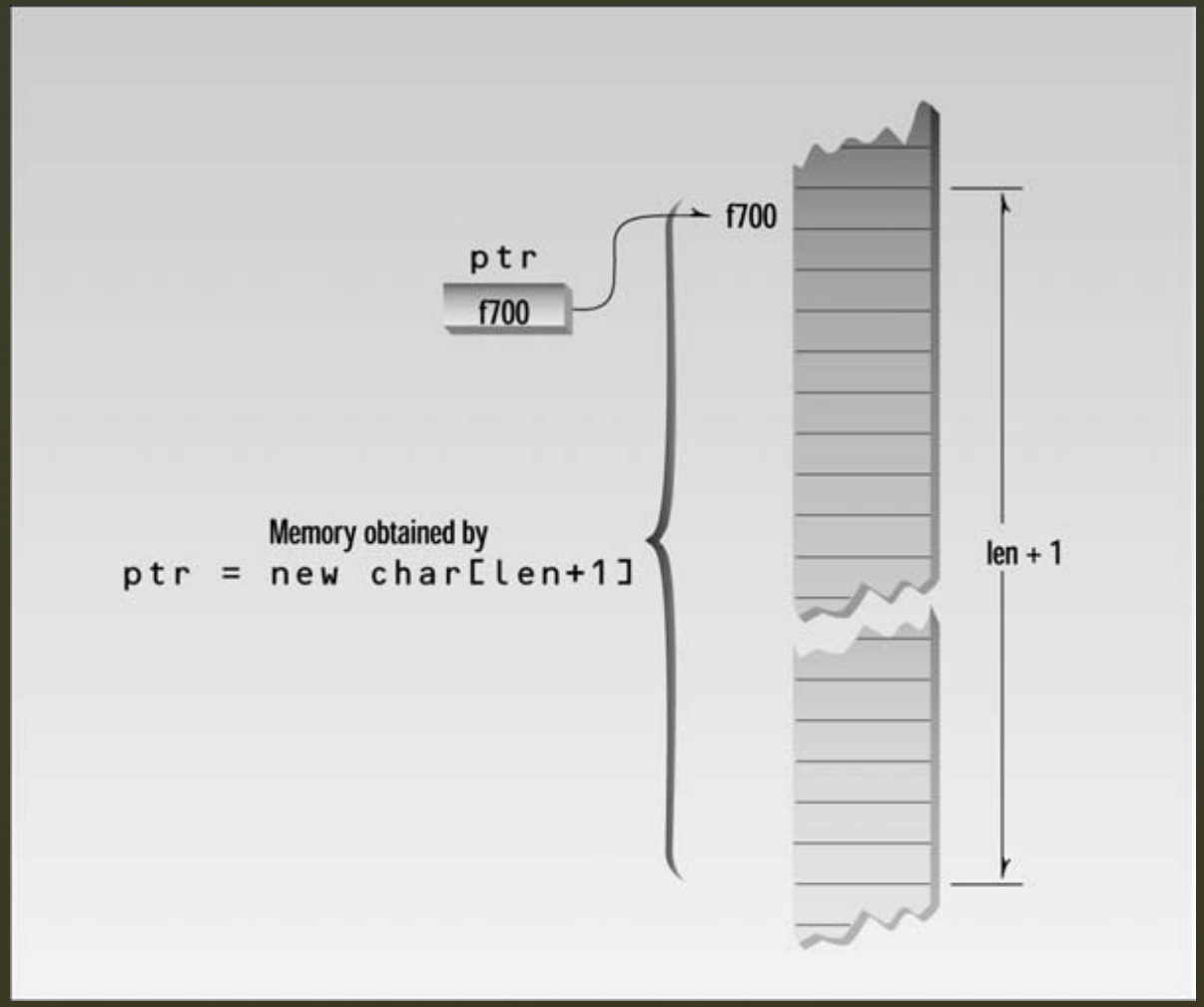
ptr = new char[len+1]; returns a pointer to a section of memory just large enough to hold the string str, whose length len can be found with the strlen() library function, plus an extra byte for the null character '\0' at the end of the string.

```
#include <iostream>
#include <cstring> //for strlen
using namespace std;

int main()
{
    const char* str = "Idle hands are the devil's workshop.";
    int len = strlen(str); //get length of str

    char* ptr; //make a pointer to char
    ptr = new char[len+1]; //set aside memory: string + '\0'

    strcpy(ptr, str); //copy str to new memory area ptr

    cout << "ptr=" << ptr << endl; //show that ptr is now in str

    delete[] ptr; //release ptr's memory
    return 0;
}
```

The new Operator



The new Operator



The delete Operator

- If your program reserves many chunks of memory using new, eventually all the available memory will be reserved and the system will crash.
- To ensure safe and efficient use of memory, the new operator is matched by a corresponding delete operator that returns memory to the operating system.
- Deleting the memory doesn’t delete the pointer that points to it and doesn’t change the address value in the pointer.
- However, this address is no longer valid; the memory it points to may be changed to something entirely different.

Pointers to Objects

Pointers can point to objects as well as to simple data types and arrays.

```
class Distance //English Distance class
{
    private:
        int feet;
        float inches;
    public:
        void getdist() //get length from user
        {
            cout << "\nEnter feet: "; cin >> feet;
            cout << "Enter inches: "; cin >> inches;
        }
        void showdist() //display distance
        { cout << feet << "\'-" << inches << '\"'; }
};

int main()
{
    Distance dist; //define a named Distance object
    dist.getdist(); //access object members
    dist.showdist(); // with dot operator

    Distance* distptr; //pointer to Distance
    distptr = new Distance; //points to new Distance object
    distptr->getdist(); //access object members
    distptr->showdist(); // with -> operator
    cout << endl;
    return 0;
}
```

An Array of Pointers to Objects

- A common programming construction is an array of pointers to objects.
- This arrangement allows easy access to a group of objects, and is more flexible than placing the objects themselves in an array.

An Array of Pointers to Objects

```
// array of pointers to objects
#include <iostream>
using namespace std;

class person //class of persons
{
    protected:
        char name[40]; //person's name
    public:
        void setName() //set the name
        {
            cout << "Enter name: ";
            cin >> name;
        }
        void printName() //get the name
        {
            cout << "\n Name is: " << name;
        }
};
```

```
int main()
    {
    person* persPtr[100]; //array of pointers to persons
    int n = 0; //number of persons in array
    char choice;

    do //put persons in array
        {
            persPtr[n] = new person; //make new object
            persPtr[n]->setName(); //set person's name
            n++; //count new person
            cout << "Enter another (y/n)? "; //enter another
            cin >> choice; //person?
        }
    while( choice=='y' ); //quit on 'n'

    for(int j=0; j<n; j++) //print names of all persons
    {
        cout << "\nPerson number " << j+1;
        persPtr[j]->printName();
    }
cout << endl;
return 0;
} //end main()
```

Exercise

- Write a program that reads a group of numbers from the user and places them in an array of type float. Once the numbers are stored in the array, the program should average them and print the result. Use pointer notation wherever possible.

Any Questions?

The End

Contact: tsadjaidoo@knust.edu.gh

Office: Caesar Building, Room 413