COMP301
Data Structures & Algorithms [C++]

Dr. N. B. Gyan

Central University, Miotso. Ghana

Arrays of Pointers

Arrays of Pointers

- Pointers offer various possibilities for simple and efficient handling of large amounts of data.
- For example, when you are sorting objects it makes sense to define pointers to those objects and simply place the pointers in order, instead of rearranging the actual order of the objects in memory.

Defining Arrays of Pointers

- Whenever you need a large number of pointers, you can define an array whose elements are pointers.
- An array of this type is referred to as a *pointer array*. E.g.

```
Account* accPtr[5];
```

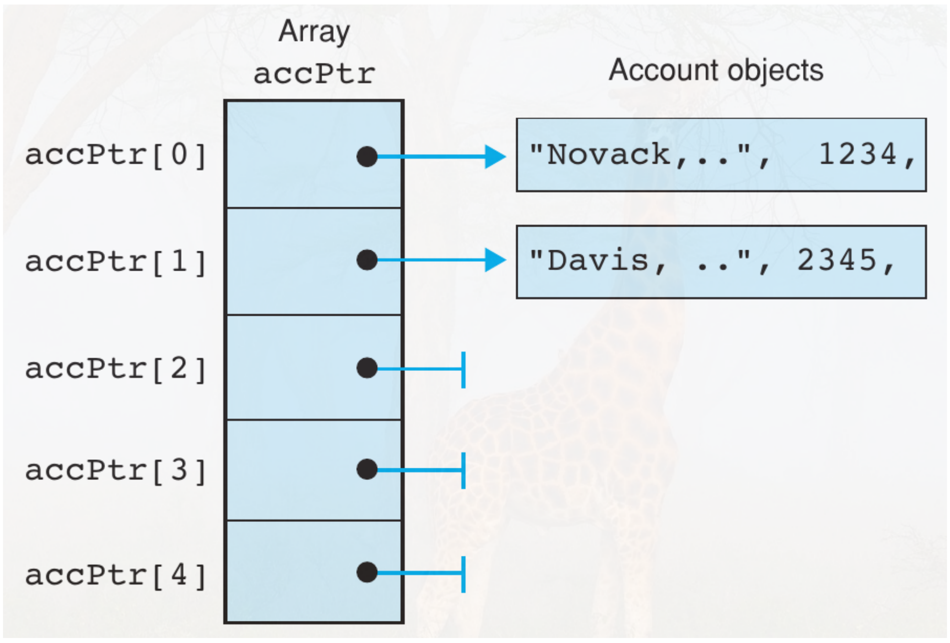
- The array `accPtr` contains five Account pointers `accPtr[0]`, `accPtr[1]`, ..., `accPtr[4]`.
- The individual pointers in the array can now be assigned object addresses.

Defining Arrays of Pointers

Any pointers not currently in use should have the value `NULL`.
E.g.

```
Account save(‘‘Novack, Kim’’,1234,9999.90);
Account depo(‘‘Davis, Sammy’’,2345,1000);
accPtr[0] = &save;
accPtr[1] = &depo;
for( int i=2; i<5; ++i) accPtr[i] = NULL;
```

Defining Arrays of Pointers



Initialisation

- As usual, an initialization list is used to initialize the array.
- In the case of a pointer array, the list contains either valid addresses or the value `NULL`. E.g.

```
Account* accPtr[5] = { &depo, &save, NULL};
```

- The value NULL is automatically assigned to any objects for which the list does not contain a value.
- This produces the same result as in the previous example.

Usage

- The individual objects addressed by the pointers in an array do not need to occupy a contiguous memory space.
- Normally these objects will be created and possibly destroyed dynamically at run-time.
- This allows for extremely flexible object handling.
- The order is defined only by the pointers.

Example usage

```
for( int i=0; i<5; ++i)
  if( accPtr[i] != NULL)
    accPtr[i]->display(); // To output
```

Command Line Arguments

- When you launch a program, you can use the command line to supply additional character sequences other than the program name.
- These *command line arguments* are typically used to govern how a program is executed or to supply the data a program will work with. E.g.

```
copy file1 file2
```

- In this case, the program copy is launched with the arguments `file1` and `file2`.

Command Line Arguments

- The individual arguments are separated by spaces.
- Characters used for redirecting input and output (`>` or `<`) and a following word are evaluated by the operating system and not passed to the program.
- If an argument contains space or redirection characters, you must place it in double quotes.

Parameters of the Function `main()`

- So far we have only used the function `main()` without parameters.
- However, if you intend to process command line arguments, you must define parameters for `main()`.

```
int main( int argc, char* argv[] )
{
  ... // Function block
}
```

- `argc` contains the number of arguments passed via the command line. The program name is one of these, so `argc` will have a value of at least 1.

Command Line Arguments

Illustrating the use of aggregates with structures: `hello.cpp`

Parameters of the Function `main()`

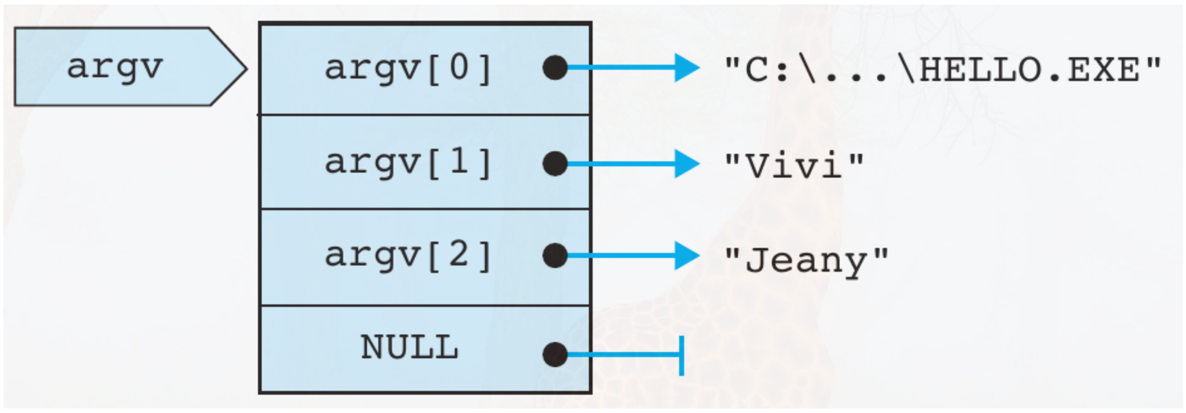
The parameter `argv` is an array of char pointers:

`argv[0]`	points to the program name (and path)
`argv[1]`	points to the first real argument, that is, the word after the program name
…	
`argv[2]`	points to the second argument
`argv[argc-1]`	points to the last argument
`argv[argc]`	is the NULL pointer

Parameters of the Function `main()`

- The parameters are traditionally named `argc` and `argv` although any other name could be used.
- Various operating systems, for example WINDOWS 98/00/NT and UNIX, allow you to declare a third parameter for `main()`.
- This parameter is an array with pointers to environment strings.

Parameters of the Function `main()`



Structures & Pointers

Structures

Illustrating the use of aggregates with structures: `structStudent2.cpp`

Pointers to Structures

Suppose that we have

```
Student s;
Student *ptr = &s;    // ptr points at
                      // structure s
```

Then we can access the grade point average by `(*ptr).gradePointAvg`.

Pointers to Structures

- The parentheses are absolutely necessary because the member operator, being a postfix operator, has higher precedence than the prefix dereferencing operator.
- The parentheses become annoying after awhile, so `C++` provides an additional postfix operator, the `->` operator, which accesses members of a pointed-at structure.
- Thus `ptr->gradePointAvg` gives the same access as before.

See you next week, God willing 🙏🏿