CS11600: Introduction to Computer Programming (C++)

Lecture 5

Svetlozar Nestorov
University of Chicago

Outline

- Computer memory
- Lvalues and rvalues
- Arrays and strings
- Pointers
- Dynamic memory allocation

Memory

- Hierarchical memory organization:
  - Cache
  - RAM (main memory)
  - Hard disk (secondary storage)
  - Tape (tertiary storage)
- Our focus is on RAM:
  - Think of it as a long list of bytes.

Heap and Stack

- The Stack:
  - When a function is called a new frame is pushed on the stack.
  - The frame contains parameters, local variables, and other info.
  - When a function call returns its frame is popped off the stack.
- The Heap:
  - For dynamically allocated memory.
  - Heap and stack are on opposite end of memory and grow towards each other.

Lvalues and Rvalues

- Lvalue is writable memory location, i.e. can be assigned a value.
- Rvalue is data at memory location.
- Constants (constant variables and literal constants) have only rvalues.
- Variables have rvalues and lvalues.
  - Lvalue is used on the left side of assignments.
  - Rvalue is used on the right side of assignments.

Arrays

- Basic form:
  
  ```
  Type name[size] = {val1, val2, ...};
  ```

  Values are optional; the number of values must be less than size but not more.
- Size must be a constant integer expression.
- Examples:
  ```
  int scores[20];
  float gpa[] = {3.4, 3.6, 2.1, 4.0}
  double prices[5] = {199.99, 201.11, 11.0}
  ```
Multidimensional Arrays

Type name[size1][size2]… = {{val1,val2, ...},...};

Example:

```c
int grades[3][4] = {
    {10, 10, 10},
    {1, 1}
};
char hi[2][2][2] = {{{'h','e'}, {'l','l'}},
    {}};
```

Accessing Arrays

- Access subscripts mimic array definition:
  ```c
type name[expr1][expr2]...
  ```
- The index expressions may involve variables and must evaluate to integers.
- Array subscripts start from 0!
- Examples:
  ```c
gpas[0] is 3.4; gpas[4] is undefined
grades[1][0] is 1; grades[2][2] is undefined
hi[0][1][0] is ?
```

Strings

- Strings are represented as NULL-terminated one-dimensional arrays of char's.
- Examples:
  ```c
  char hello[] = "hello"; is equivalent to
  char hello[] = {'h','e','l','l','o','\0'};
  ```

Pointers

- A pointer is a memory address:
  ```c
  Type *pname = value;
  ```
- Accessing data pointed to by a pointer is called dereferencing:
  ```c
  *pname
  ```
- A pointer definition does not allocate memory for the data to which it points!
- A pointer can be initialized with a reference to already defined variable of the appropriate type.
- Examples

Pointers and Arrays

- Pointers and array are related by the following rule:
  ```c
  name[i] is equivalent to *(name + i)
  ```
- Example:
  ```c
  char hi = "hello";
  *hi is 'h'; *(hi+4) is 'o';
  ```

Dynamic Memory Allocation

- Why do need it?
- Two operators:
  ```c
  *new allocates memory.
  *delete de-allocates memory previously allocated with new.
  ```
- Memory is allocated on the heap.
- No garbage collection – delete what you allocated!
New

- Basic form:
  ```cpp
ew Type
```
- With initialization
  ```cpp
  new Type(value)
  ```
- Returns a pointer to an object of `Type`.
- Example:
  ```cpp
  int *n = new int(5);
  char *p;
  p = new char;
  ```

New and Arrays

- Primary use of new is for allocating arrays of variable length and user-defined types.
- Syntax mimics array declaration:
  ```cpp
  new Type[size1][size2]...
  ```
- `size1` can be a variable expression.
- Returns a pointer to the first element.
  ```cpp
  But memory is allocated for all elements!
  ```
- Examples.

Delete

- De-allocates memory allocated with `new`
  ```cpp
  delete ptr;
  delete [] ptr; (for arrays)
  ```
- Example:
  ```cpp
  int *zips = new int[k];
  zips[0] = 60611;
  /* do something with zips */
  delete [] zips;
  ```