CS 300 Data Structures

Fall 2021

Dr. Fatma Cemile Serce
Dr. Neelakantan Kartha
Dr. Z li

1

Week-1

- Introduction
- Statements
- Functions
- Arrays
- Structures
- Introduction to Pointers

2

Introduction

Compiled Languages and C++

Why C++

History of C++

Hello World

Syntax and Data types

3

C++ Programming language

- C++ was developed by Bjarne Stroustrup at Bell Laboratories , 1979
  - Originally called “C with classes”
  - The name C++ is based on C’s increment operator (++)
    - indicating that C++ is an enhanced version of C
- A C++ program is a collection of one or more subprograms (functions)
- The key concept in C++ is class. A class is a user-defined type
- C++ and its standard libraries are designed for portability. The current implementation will run on most systems that support C.
- C libraries can be used from a C++ program, and most tools that support programming in C can be used with C++.

https://www.youtube.com/watch?v=JBjjnqG0BP8

Bjarne Stroustrup: Why I Created C++

4

4

Who is this computer scientist?

Bjarne Stroustrup

- 2002-2014: Chair of CS at Texas A&M
- Currently
  - Managing director at Morgan Stanley
  - Visiting professor at Columbia
- Invented C++
- Wrote "The C++ programming Language"
- Still deeply engaged in language

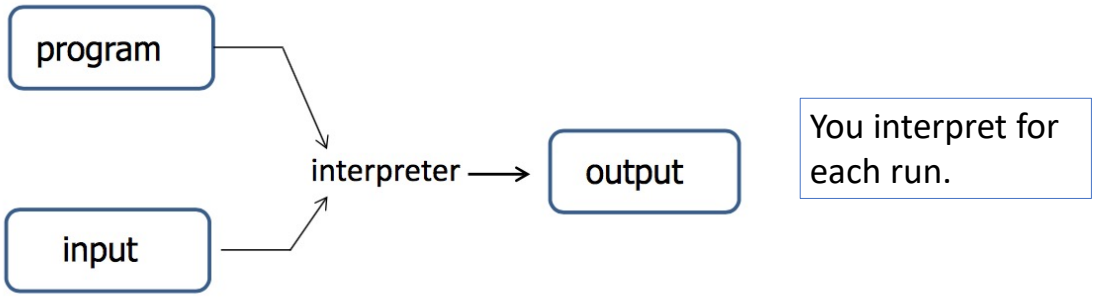
5

Program Execution

- Interpretation
  - Interpreter: a program that executes program statements
  - Directly interprets program
  - Limited optimization
  - PHP, LISP, Python, Mathlab
- Compilation
  - Compiler: translates statements into machine language
  - Creates executable
  - Performs optimization over multiple statements
  - C, C++
- Hybrid
  - Java has features of both

6

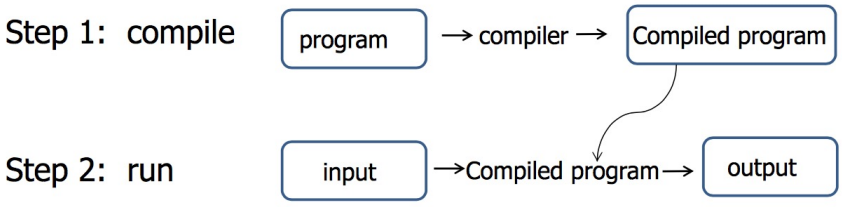
Interpretation



7

Compilation

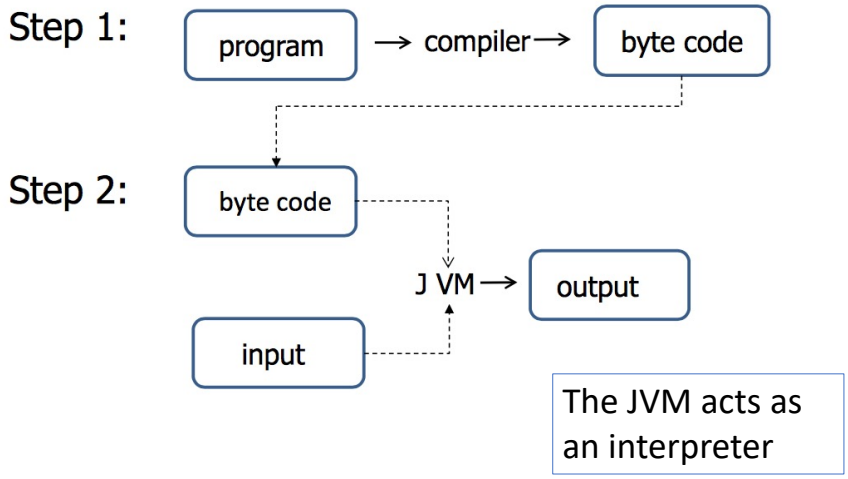
Compilation diagram



You compile once and run many times.

8

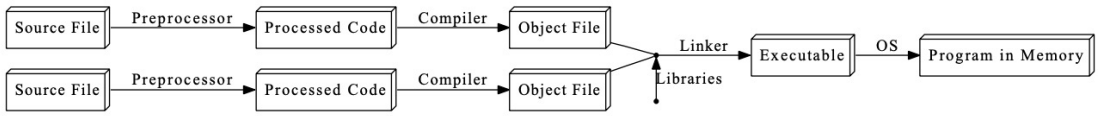
Hybrid: Java case



9

C++ Compilation Process

- C++ adds an extra step to the compilation process:
  - the code is run through a preprocessor, which applies some modifications to the source code, before being fed to the compiler



10

Why C++?

- Maintainability
  - Object-oriented
- Portability
  - one of C++'s strengths is that it can be used to write programs for nearly any processor.
- Efficient
- Some nice features
  - Type Safety
  - Operator Overloading
- Provides system understanding
  - Memory (stack/heap),
  - Usage of pointers / memory allocation /deallocation
  - C++ does give access to some lower-level functionality than other languages (e.g. memory addresses)

11

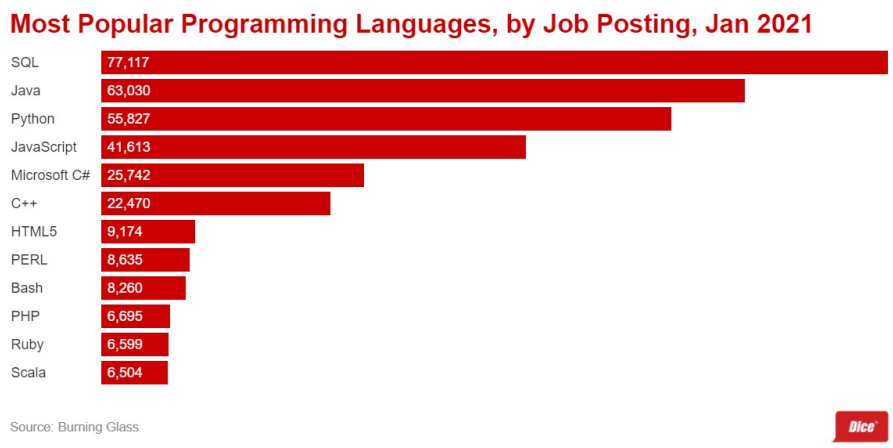
Why C++?

Sep 2021	Sep 2020	Change	Programming Language	Ratings	Change
1	1		C	11.83%	-4.12%
2	3	↑	Python	11.67%	+1.20%
3	2	↓	Java	11.12%	-2.37%
4	4		C++	7.13%	+0.01%
5	5		C#	5.78%	+1.20%
6	6		Visual Basic	4.62%	+0.50%
7	7		JavaScript	2.55%	+0.01%
8	14	⇈	Assembly language	2.42%	+1.12%

Source:
https://www.tiobe.com/tiobe-index/

12

Why C++?



13

Why C++?

Total

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67
(v) Racket	11.27
(i) Hack	26.99
(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90
(i) Lua	82.91

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

Efficiency: Computer Benchmark Game

14

Some C++ history…

- 1979: “C with Classes”
- 1983: Name changed to C++
- 1985: *The C++ Programming Language* published
- 1990: Turbo C++ released (significant library support)
- 1998: C++ ISO/IEC 14882:1998 published (with STL defined)
- 2011: C++11 standard published
- 2017: C++17 standard

15

“Hello World!”

16

"Hello World"

```
//Hello World
#include <iostream>

using namespace std;

int main()
{
   cout << "Hello World!\n" << endl;
   return 0;
}
```

17

Dissecting "Hello World"

- **//**: indicates that everything following it until the end of the line is a comment: it is ignored by the compiler.
- #: preprocessor commands
- **#include:** tells the preprocessor to dump in the contents of another file, here the iostream file
- **<iostream>:** defines the procedures for input/output (header file)
- **using namespace std**: In C++, identifiers can be defined within a context – sort of a directory of names – called a namespace. This statement tells the compiler that it should look in the std namespace for any identifier we haven't defined.
- **main:** defines the code that should execute when the program starts up.
- **std::cout** : output stream, outputting some piece of text to the screen.
- **<<:** operator overloaded
- **"Hello World":** string literal
- **"\n":** escape character for a line break
- **return 0:** indicates that the program should tell the operating system it has completed successfully

18

C++ Data Types

Name	Description	Size*	Range*
`char`	Character or small integer.	1byte	signed: -128 to 127 unsigned: 0 to 255
`short int` (`short`)	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
`int`	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
`long int` (`long`)	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
`bool`	Boolean value. It can take one of two values: true or false.	1byte	true or false
`float`	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
`double`	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
`long double`	Long double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)

19

Exercise

- `75  // ?`
- `75u  // ?`
- `75l  // ?`
- `75ul  // ?`

20

Exercise

```
#include <iostream>
using namespace std;
namespace n1
{
    int x = 9;
    float y = 3.1;
}
namespace n2
{
    double x = 29;
    char y = 'a';
}
int main ()
{
    using namespace n1;
    cout<< x << endl;
    cout<< y << endl;
    cout<< n2::x << endl;
    cout<< n2::y << endl;
    return 0;
}
```

21

Input

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

const double pi = 3.14159;

int main()
{
        double radius;

        cout << “Input a non-negative number for the radius of a circle: ”;
        cin >> radius;
        cout << “The area of the circle if radius =  " << radius <<  “is”;
       cout << pi*radius*radius << endl;
}
```

Variables can be "cascaded"

`cin >> amount >> count >> direction;`

22

cin

- During execution of a cin command
  - as long as it keeps finding data, it keeps reading
  - when the reading marker hits something **not** data, it **quits** reading
- Things in the input stream that cin considers not data
  - spaces
  - tab \t
  - newline character \n
    (pressing the RETURN key)
  - for numeric input, something nonnumeric

23

cin and strings

- In order to get entire lines, we can use the function `getline`

```
#include <iostream>
using namespace std;
int main () {
  string message;
  cout<<"Please enter a message:";
  getline(cin, message);
  cout<<"Message:"<<message;
  return 0;
}
```

24

24

stringstream

- The standard header file <sstream> defines a class called stringstream that allows a string-based object to be treated as a stream.

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;
int main () {
  string mystr;
  float price=0;
  int quantity=0;
  cout << "Enter price: ";
  getline (cin,mystr);
  stringstream(mystr) >> price;
  cout << "Enter quantity: ";
  getline (cin,mystr);
  stringstream(mystr) >> quantity;
  cout << "Total price: " << price*quantity <<endl;
  return 0;
}
```

```
Enter price: 12.56
Enter quantity: 12
Total price: 150.72
```

25

25

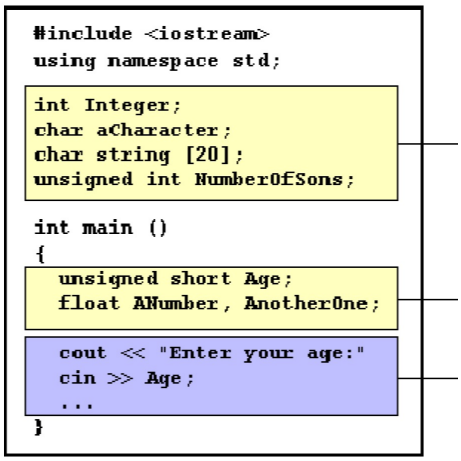
Scope

```
#include <iostream>
using namespace std;
int main () {
  int i=90;
  int sum = 0;
  for(int i=0;i<5;i++)
  {
       cout<<i<<" ";
       {
           sum += i;
           cout<<sum<<endl;
       }
  }
  cout<<i;
}
```

Output: ?

26

Scope: Global and Local Variables



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

int Integer;
char aCharacter;
char string [20];
unsigned int NumberOfSons;

int main ()
{
  unsigned short Age;
  float ANumber, AnotherOne;

  cout << "Enter your age:"
  cin >> Age;
  ...
}
```

Global variable: Global variables can be referred from anywhere in the code, even inside functions, whenever it is after its declaration.

Local variable

27

27

Operators

- Arithmetic operators (+, -, *, /, %)
- Compound assignment (+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)
- Increment and decrement operators (++, --)
- Relational and equality operators (==, !=, >, <, >=, <=)
- Logical operators (!, &&, ||)
- Bitwise Operators (&, |, ^, ~, <<, >>)
- Comma operator (,) : to separate two or more expressions that are included where only one expression is expected
- Conditional operator (?)

```
condition ? result1 : result2
```

28

Exercise

```
#include <iostream>
using namespace std;
int main () {
  int a,b,c;
  a=2;
  b=7;
  a = (b=3, b+2);
  cout << a <<" "<<b;
  return 0;
}
```

Output: ?

29

Statements

30

Statements

- Conditionals
- Loops
- break, continue
- Compound statements

31

Exercise

```
#include <iostream>
int main () {
    int x = 50;
    if(x>90)
       if(x>80)
          std::cout<<x;
    else
       std::cout<<"less than 60";
    return 0;
}
```

Output: ?

32

Exercise

```
#include <iostream>
int main () {
    int x = 5;
    if ( x = 10) {
      std::cout<<“ > 10”;
    } else{
      std::cout<<“ < 10”;
    }
}
```

Output: ?

33

Loops

- for loop

```
for(i=0; i<n; i++)
{
   cout << A[i]<<endl;
}

for(;;) {…}
```

- while loop

```
while (i>10) { x-=4;i--;}
```

- do-while loop

```
do {x -=4;i--} while (i>10);
```

34

break and continue

```
for (; ;){
  …
  if (a==b) break;
  …
}
-----------------------------------------
for (;;){
  …
  if (a==b) continue;
  …
}
```

35

Functions

36

Functions in C++

- Functions defined in the standard library
- Self defined-functions
- Value-Returning Functions
- Function Definition
- Function Prototype
- Flow of Execution

37

Sample Functions in the Standard Library

Function	Standard Header File	Purpose	Parameter(s) Type	Result
`abs(x)`	`<cstdlib>`	Returns the absolute value of its argument: `abs(-7) = 7`	`int`	`int`
`ceil(x)`	`<cmath>`	Returns the smallest whole number that is not less than x: `ceil(56.34) = 57.0`	`double`	`double`
`cos(x)`	`<cmath>`	Returns the cosine of angle x: `cos(0.0) = 1.0`	`double` (radians)	`double`
`exp(x)`	`<cmath>`	Returns e^x, where `e = 2.718`: `exp(1.0) = 2.71828`	`double`	`double`
`fabs(x)`	`<cmath>`	Returns the absolute value of its argument: `fabs(-5.67) = 5.67`	`double`	`double`
`floor(x)`	`<cmath>`	Returns the largest whole number that is not greater than x: `floor(45.67) = 45.00`	`double`	`double`
`pow(x,y)`	`<cmath>`	Returns x^y; if x is negative, y must be a whole number: `pow(0.16, 0.5) = 0.4`	`double`	`double`
`tolower(x)`	`<cctype>`	Returns the lowercase value of x if x is uppercase; otherwise, returns x	`int`	`int`
`toupper(x)`	`<cctype>`	Returns the uppercase value of x if x is lowercase; otherwise, returns x	`int`	`int`

38

Programmer-defined Functions

- Value returning function
  - computes a single value
  - returns value to calling code
  - uses `return` command
- Void function (procedure)
  - called as a statement
  - executes some task

39

Function Definition Syntax

```
functionType functionName (<formal parameter list>)
{
     statements
}

double circleArea(int r){
    return 3.14*r*r;
}
```

40

Function Call Syntax

functionName (params);

- ```
  cout <<"Enter radius for circle area -> ";
  cin >> radius;
  area = circleArea (radius);
  ```

41

Function Prototype

- To call a function it must have been declared in some earlier point of the code
- There is an alternative way to avoid writing the whole code of a function before it can be used in main or in some other function.
- By declaring just a prototype of the function before it is used, instead of the entire definition.
- Function Prototype Syntax:

  <type> <function name>(<type list>);

42
  ```

```
// declaring functions prototypes
#include <iostream>
using namespace std;
void odd (int a); //function prototype
void even (int a); //function prototype
int main () {
   int i;
   do {
      cout << "Type a number (0 to exit): ";
      cin >> i;
      odd (i);
   } while (i!=0);
   return 0;
}
void odd (int a) {//function definition
   if ((a%2)!=0)
      cout << "Number is odd.\n";
   else
      even (a);
}
void even (int a) {//function definition
   if ((a%2)==0)
      cout << "Number is even.\n";
   else
      odd (a);
}
```

- include statements
- function prototypes
- main function
- function definitions

43

Reference Parameters

- Synonyms of objects they reference
  - Reference are not pointers
- Avoid the cost of copying
- E.g.

```
string x = findMax(a);
string &y  = x;
cout << y << endl;
```

44

Example: Reference Parameters

```
void swap(int& x, int& y) {
    int temp = x; // temp is a local variable
    x = y; // changes the actual parameter in the calling pgm.
    y = temp; // changes the actual parameter in the calling pgm.
}
int main(){
    int x=10, y=60;
    int &a = x;
    a = 15;
    int &b = y;
    swap(a,b);
    cout<<a<<“ “<<b;
}
```

Output:?

45

Default Paramaters

- Default parameters are used in place of the missing trailing arguments in a function call

```
#include <iostream>
using namespace std;
void m(int x=1,int y=20)   // default parameters
{
    cout<<x<<" "<<y<<endl;
}
int main()
{
    m(5,10);
    m(5);
    m();
    return 0;
}
```

46

Arrays

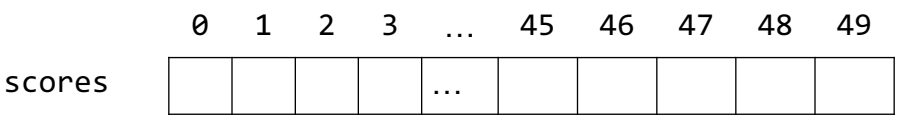
47

Arrays

- Compound Data Types
- a series of elements of the same type placed in contiguous memory locations
- fized size
- individually referenced by adding an index to a unique identifier.

Example:

```
int scores[50]; //single-dimensional
```



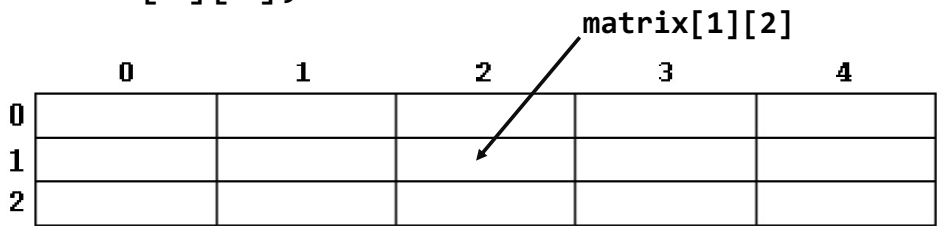
48

Multidimensional Arrays

- Arrays of arrays

Example:

```
int matrix[3][5];
```



49

Example: Multidimensional Arrays

```
#include <iostream>
#include <fstream>
using namespace std;
#define WIDTH 5
#define HEIGHT 3
int matrix[HEIGHT][WIDTH];

int main () {
    int n,m;
    for (n=0;n<HEIGHT;n++)
       for (m=0;m<WIDTH;m++)
           matrix[n][m]=(n+1)*(m+1);

    for (n=0;n<HEIGHT;n++){
       for (m=0;m<WIDTH;m++)
           cout<<matrix[n][m]<<" ";
       cout<<endl;
    }
    return 0;
}
```

```
1 2 3 4 5
2 4 6 8 10
3 6 9 12 15
```

50

Arrays as Parameters

```
#include <iostream>
using namespace std;
void printarray (int arg[], int length) {
  for (int n=0; n<length; n++)
    cout<<arg[n]<<" ";
  cout<<"\n";
}
int main () {
  int firstarray[] = {5, 10, 15};
  int secondarray[] = {2, 4, 6, 8, 10};
  printarray (firstarray,3);
  printarray (secondarray,5);
  return 0;
}
```

51

Arrays as Parameters

```
float add(float[]); //function prototype
float add(float par[]) //function definition
```
or

```
float add(float[4]); //function prototype
float add(float par[4]) //function definition
```

void procedure (int matrix[][4]) //have to pass column value, **why?**

52

Structures

53

Structures

The first step in building a new type is often to organize the elements it needs into a data structure: **a struct**

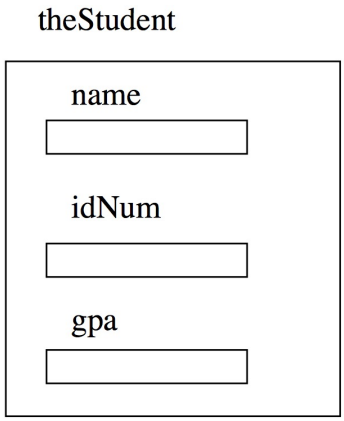
```
struct name{
     type member1;
     type member2;
         …
};
```

54

Example: Structures

```
struct StudentRec {
        string name;
        string idNum;
        float gpa;
};

StudentRec theStudent;
theStudent.name = "Sally";
cin>> theStudent.idNum;
cout<< theStudent.gpa;
```



55

Example: Structures

```
#include <iostream>
using namespace std;
struct rational
{
  int numerator;
  int denominator;
};
int main()
{
    rational n1,n2,result;
    cout<<"Please numerator and denominator for first number:";
    cin>>n1.numerator>>n1.denominator;
    cout<<"Please numerator and denominator for second number:";
    cin>>n2.numerator>>n2.denominator;
    result.numerator=n1.numerator* n2.numerator;
    result.denominator=n1.denominator* n2.denominator;
    cout<< n1.numerator<<"/"<< n1.denominator<<" x ";
    cout<< n2.numerator<<"/"<< n2.denominator<<" = ";
    cout<<result.numerator<<"/"<<result.denominator<<endl;
    return 0;
}
```

```
Please numerator and denominator for first number:10 3
Please numerator and denominator for second number:2 6
10/3 x 2/6 = 20/18
```

56

Nested Structures

```
struct address
{
    string city;
    string state;
    string zipcode;
};
struct employee
{
    int id;
    string name;
    float salary;
    struct address add;
};
```

```
void print(employee);
int main()
{
    struct employee e;
    e.id = 123;
    e.name = "John";
    e.salary=60.000;
    e.add.city = "Redmond";
    e.add.state = "WA";
    e.add.zipcode = "98052";
    print(e);
}
void print(employee e){
    cout<<e.id<<" "<<e.name<<" "<<e.add.state;
}
```

57

Structures as Function Parameters

```
#include <iostream>
using namespace std;
struct rational
{
  int numerator;
  int denominator;
};
void printRational(rational);
int main()
{
    rational number;
    number.numerator = 5;
    number.denominator = 10;
    printRational(number);
    return 0;
}
void printRational(rational number){
    cout<< number.numerator<<"/"<< number.denominator;
}
```

Output
5/10

58

Introduction to Pointers

59

Memory cell and addresses

- The single-byte memory cells
- numbered in a consecutive way
- every cell has the same number as the previous one plus one
  - Example: for cell 1776
    - it is going to be right between cells 1775 and 1777,
    - exactly one thousand cells after 776
    - exactly one thousand cells before cell 2776

60

Reference operator (&)

- Reference variable: the address that locates a variable within memory
- can be obtained by preceding the identifier of a variable with an ampersand sign (&), known as reference operator (translated as "address of".
- For example:
  x = &y;
  - This would assign to **x** the address of variable **y**.

61

Pointer

- The variable that stores the reference to another variable (like x) is what we call a **pointer**.

  x = &y; //x is a pointer

- Pointers are said to "point to" the variable whose reference they store.

62

Dereference operator (*)

- In order to directly access the value stored in the variable which it points to, precede the pointer's identifier with an asterisk (*)
- acts as dereference operator (translated to "value pointed by")
- Example:

```
t = *x;     // t equal to the value pointed by x
```

63

& vs *

- & is the reference operator and can be read as "address of"
- * is the dereference operator and can be read as "value pointed by"

- A variable referenced with & can be dereferenced with *
- Example: Assume that **y** is placed during runtime in the memory address 1776 and initial value of **y** is 50.

```
x = &y;
*x == 50;
x == 1776
&y == 1776
```

64

Declaring variables of pointer types

- Syntax:

```
type* name;
```

- where type is the data type of the value that the pointer is intended to point to
- type is not the type of the pointer itself! but the type of the data the pointer points to
- Example:

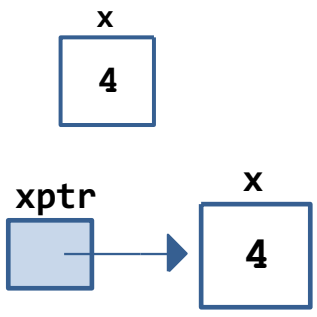
```
int *xptr;
int*  xptr;
float  *a, *b, *c;   //what if float *a,b,c; ???
```

65

Declaring variables of pointer types

```
int x = 4;
```

```
int* xptr = &x;
```



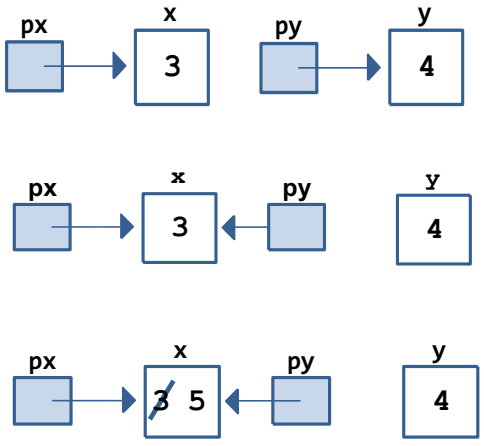
66

Example

```
int *px, *py;
int x=3, y=4;
px = &x;
py = &y;

py = px;


*py = 5;
cout<<*px;
```



67