

Introduction to C++

with content from www.cplusplus.com

Introduction

- C++
 - widely-used general-purpose programming language
 - compiled
 - procedural and object-oriented support
 - strong library support
 - created by Bjarne Stroustrup starting in 1979
 - based on C
 - first called “C with Classes”
 - also with inheritance, inlining, default function arguments, and strong type checking
 - many C programs compile with C++ compiler
 - major releases in 1983, 1989, 1998, 2011 (C++11)

Structure of a C++ Program

```
1 // my first program in C++
2 #include <iostream>
3
4 int main()
5 {
6     std::cout << "Hello World!";
7 }
```

Hello World!

Structure of a C++ Program

–previous program could also be written as follows

```
int main () { std::cout << " Hello World! "; std::cout << " I'm a C++ program "; }
```

```
1 int main ()  
2 {  
3     std::cout <<  
4         "Hello World!";  
5     std::cout  
6         << "I'm a C++ program";  
7 }
```

Structure of a C++ Program

–two styles of comments

```
1 /* my second program in C++  
2    with more comments */  
3  
4 #include <iostream>  
5  
6 int main ()  
7 {  
8     std::cout << "Hello World! ";    // prints Hello World!  
9     std::cout << "I'm a C++ program"; // prints I'm a C++ program  
10 }
```

Structure of a C++ Program

- namespace

```
1 // my second program in C++
2 #include <iostream>
3 using namespace std;
4
5 int main ()
6 {
7     cout << "Hello World! ";
8     cout << "I'm a C++ program";
9 }
```

Identifiers

- similar to rules for Python identifiers
- case-sensitive
- keywords

`alignas, alignof, and, and_eq, asm, auto, bitand, bitor, bool, break, case, catch, char, char16_t, char32_t, class, compl, const, constexpr, const_cast, continue, decltype, default, delete, do, double, dynamic_cast, else, enum, explicit, export, extern, false, float, for, friend, goto, if, inline, int, long, mutable, namespace, new, noexcept, not, not_eq, nullptr, operator, or, or_eq, private, protected, public, register, reinterpret_cast, return, short, signed, sizeof, static, static_assert, static_cast, struct, switch, template, this, thread_local, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void, volatile, wchar_t, while, xor, xor_eq`

Types

Group	Type names*	Notes on size / precision
Character types	<code>char</code>	Exactly one byte in size. At least 8 bits.
	<code>char16_t</code>	Not smaller than <code>char</code> . At least 16 bits.
	<code>char32_t</code>	Not smaller than <code>char16_t</code> . At least 32 bits.
	<code>wchar_t</code>	Can represent the largest supported character set.
Integer types (signed)	<code>signed char</code>	Same size as <code>char</code> . At least 8 bits.
	<code>signed short int</code>	Not smaller than <code>char</code> . At least 16 bits.
	<code>signed int</code>	Not smaller than <code>short</code> . At least 16 bits.
	<code>signed long int</code>	Not smaller than <code>int</code> . At least 32 bits.
	<code>signed long long int</code>	Not smaller than <code>long</code> . At least 64 bits.
Integer types (unsigned)	<code>unsigned char</code>	(same size as their signed counterparts)
	<code>unsigned short int</code>	
	<code>unsigned int</code>	
	<code>unsigned long int</code>	
	<code>unsigned long long int</code>	
Floating-point types	<code>float</code>	
	<code>double</code>	Precision not less than <code>float</code>
	<code>long double</code>	Precision not less than <code>double</code>
Boolean type	<code>bool</code>	
Void type	<code>void</code>	no storage
Null pointer	<code>decltype(nullptr)</code>	

Variables

– must be declared

```
1 // operating with variables
2
3 #include <iostream>
4 using namespace std;
5
6 int main ()
7 {
8     // declaring variables:
9     int a, b;
10    int result;
11
12    // process:
13    a = 5;
14    b = 2;
15    a = a + 1;
16    result = a - b;
17
18    // print out the result:
19    cout << result;
20
21    // terminate the program:
22    return 0;
23 }
```

Initializing Variables

- different ways to initialize variables at declaration

```
1 // initialization of variables
2
3 #include <iostream>
4 using namespace std;
5
6 int main ()
7 {
8     int a=5;           // initial value: 5
9     int b(3);          // initial value: 3
10    int c{2};           // initial value: 2
11    int result;         // initial value undetermined
12
13    a = a + b;
14    result = a - c;
15    cout << result;
16
17    return 0;
18 }
```

Variables

- automatic type deduction
- with initialization

```
1 int foo = 0;  
2 auto bar = foo; // the same as: int bar = foo;
```

- without initialization

```
1 int foo = 0;  
2 decltype(foo) bar; // the same as: int bar;
```

- used in cases where type cannot be obtained easily for generality

Strings

```
1 // my first string
2 #include <iostream>
3 #include <string>
4 using namespace std;
5
6 int main ()
7 {
8     string mystring;
9     mystring = "This is the initial string content";
10    cout << mystring << endl;
11    mystring = "This is a different string content";
12    cout << mystring << endl;
13    return 0;
14 }
```

Literals

–integers

```
1 75          // decimal
2 0113        // octal
3 0x4b        // hexadecimal
```

–floats

```
1 3.14159     // 3.14159
2 6.02e23     // 6.02 x 10^23
3 1.6e-19     // 1.6 x 10^-19
4 3.0         // 3.0
```

–chars

```
1 'z'
2 'p'
3 "Hello world"
4 "How do you do?"
```

Literals

– escape sequences

Escape code	Description
<code>\n</code>	newline
<code>\r</code>	carriage return
<code>\t</code>	tab
<code>\v</code>	vertical tab
<code>\b</code>	backspace
<code>\f</code>	form feed (page feed)
<code>\a</code>	alert (beep)
<code>\'</code>	single quote (')
<code>\"</code>	double quote (")
<code>\?</code>	question mark (?)
<code>\\</code>	backslash (\)

Constants

-typed constants

```
1 #include <iostream>
2 using namespace std;
3
4 const double pi = 3.14159;
5 const char newline = '\n';
6
7 int main ()
8 {
9     double r=5.0;           // radius
10    double circle;
11
12    circle = 2 * pi * r;
13    cout << circle;
14    cout << newline;
15 }
```

31.4159

Constants

–#define constants

```
1 #include <iostream>
2 using namespace std;
3
4 #define PI 3.14159
5 #define NEWLINE '\n'
6
7 int main ()
8 {
9     double r=5.0;           // radius
10    double circle;
11
12    circle = 2 * PI * r;
13    cout << circle;
14    cout << NEWLINE;
15
16 }
```

31.4159

Increment/Decrement

– prefix/postfix

Example 1	Example 2
<pre>x = 3; y = ++x; // x contains 4, y contains 4</pre>	<pre>x = 3; y = x++; // x contains 4, y contains 3</pre>

Operators

–if a=2, b=3, c=6

```
1 (a == 5)      // evaluates to false, since a is not equal to 5
2 (a*b >= c)    // evaluates to true, since (2*3 >= 6) is true
3 (b+4 > a*c)   // evaluates to false, since (3+4 > 2*6) is false
4 ((b=2) == a) // evaluates to true
```

–AND/OR

```
1 ( (5 == 5) && (3 > 6) ) // evaluates to false ( true && false )
2 ( (5 == 5) || (3 > 6) ) // evaluates to true ( true || false )
```

–other operators work similarly to Python

Ternary Operator

– condition ? result1 : result2

```
1 7==5 ? 4 : 3      // evaluates to 3, since 7 is not equal to 5.  
2 7==5+2 ? 4 : 3    // evaluates to 4, since 7 is equal to 5+2.  
3 5>3 ? a : b       // evaluates to the value of a, since 5 is greater than 3.  
4 a>b ? a : b       // evaluates to whichever is greater, a or b.
```

```
1 // conditional operator  
2 #include <iostream>  
3 using namespace std;  
4  
5 int main ()  
6 {  
7     int a,b,c;  
8  
9     a=2;  
10    b=7;  
11    c = (a>b) ? a : b;  
12  
13    cout << c << '\n';  
14 }
```

Bitwise Operators

operator	asm equivalent	description
&	AND	Bitwise AND
	OR	Bitwise inclusive OR
^	XOR	Bitwise exclusive OR
~	NOT	Unary complement (bit inversion)
<<	SHL	Shift bits left
>>	SHR	Shift bits right

Type Casting

–both OK

```
1 int i;  
2 float f = 3.14;  
3 i = (int) f;
```

```
i = int (f);
```

Operator Precedence

Level	Precedence group	Operator	Description	Grouping
1	Scope	::	scope qualifier	Left-to-right
2	Postfix (unary)	++ --	postfix increment / decrement	Left-to-right
		()	functional forms	
		[]	subscript	
		. ->	member access	
3	Prefix (unary)	++ --	prefix increment / decrement	Right-to-left
		~ !	bitwise NOT / logical NOT	
		+ -	unary prefix	
		& *	reference / dereference	
		new delete	allocation / deallocation	
		sizeof	parameter pack	
		(type)	C-style type-casting	
4	Pointer-to-member	.* ->*	access pointer	Left-to-right
5	Arithmetic: scaling	* / %	multiply, divide, modulo	Left-to-right
6	Arithmetic: addition	+ -	addition, subtraction	Left-to-right
7	Bitwise shift	<< >>	shift left, shift right	Left-to-right
8	Relational	< > <= >=	comparison operators	Left-to-right
9	Equality	== !=	equality / inequality	Left-to-right
10	And	&	bitwise AND	Left-to-right
11	Exclusive or	^	bitwise XOR	Left-to-right
12	Inclusive or		bitwise OR	Left-to-right
13	Conjunction	&&	logical AND	Left-to-right
14	Disjunction		logical OR	Left-to-right
15	Assignment-level expressions	= *= /= %= += -=	assignment / compound assignment	Right-to-left
		>>= <<= &= ^= =		
		?:	conditional operator	
16	Sequencing	,	comma separator	Left-to-right

Input/Output

```
1 // i/o example
2
3 #include <iostream>
4 using namespace std;
5
6 int main ()
7 {
8     int i;
9     cout << "Please enter an integer value: ";
10    cin >> i;
11    cout << "The value you entered is " << i;
12    cout << " and its double is " << i*2 << ".\n";
13    return 0;
14 }
```

```
Please enter an integer value: 702
The value you entered is 702 and its double is 1404.
```

Input/Output

```
1 // cin with strings
2 #include <iostream>
3 #include <string>
4 using namespace std;
5
6 int main ()
7 {
8     string mystr;
9     cout << "What's your name? ";
10    getline (cin, mystr);
11    cout << "Hello " << mystr << ".\n";
12    cout << "What is your favorite team? ";
13    getline (cin, mystr);
14    cout << "I like " << mystr << " too!\n";
15    return 0;
16 }
```

```
What's your name? Homer Simpson
Hello Homer Simpson.
What is your favorite team? The Isotopes
I like The Isotopes too!
```


if Statements

– compound if

```
1 if (x > 0)
2     cout << "x is positive";
3 else if (x < 0)
4     cout << "x is negative";
5 else
6     cout << "x is 0";
```

Iteration

– while statement

```
1 // custom countdown using while
2 #include <iostream>
3 using namespace std;
4
5 int main ()
6 {
7     int n = 10;
8
9     while (n>0) {
10         cout << n << ", ";
11         --n;
12     }
13
14     cout << "liftoff!\n";
15 }
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, liftoff!
```

Iteration

-do-while statement

```
1 // echo machine
2 #include <iostream>
3 #include <string>
4 using namespace std;
5
6 int main ()
7 {
8     string str;
9     do {
10         cout << "Enter text: ";
11         getline (cin, str);
12         cout << "You entered: " << str << '\n';
13     } while (str != "goodbye");
14 }
```

```
Enter text: hello
You entered: hello
Enter text: who's there?
You entered: who's there?
Enter text: goodbye
You entered: goodbye
```

Iteration

–for loop

```
1 // countdown using a for loop
2 #include <iostream>
3 using namespace std;
4
5 int main ()
6 {
7     for (int n=10; n>0; n--) {
8         cout << n << ", ";
9     }
10    cout << "liftoff!\n";
11 }
```

10, 9, 8, 7, 6, 5, 4, 3, 2, 1, liftoff!

Diagram illustrating the components of a for loop:

```
for ( n=0, i=100 ; n!=i ; ++n, --i )
```

- Initialization: `n=0, i=100`
- Condition: `n!=i`
- Increase: `++n, --i`

Iteration

– range-based for loop

```
1 // range-based for loop
2 #include <iostream>
3 #include <string>
4 using namespace std;
5
6 int main ()
7 {
8     string str {"Hello!"};
9     for (char c : str)
10    {
11        std::cout << "[" << c << " ";
12    }
13    std::cout << '\n';
14 }
```

```
[H] [e] [l] [l] [o] [!]
```

break Statement

-break

```
1 // break loop example
2 #include <iostream>
3 using namespace std;
4
5 int main ()
6 {
7     for (int n=10; n>0; n--)
8     {
9         cout << n << ", ";
10        if (n==3)
11        {
12            cout << "countdown aborted!";
13            break;
14        }
15    }
16 }
```

10, 9, 8, 7, 6, 5, 4, 3, countdown aborted!

continue Statement

–continue

```
1 // continue loop example
2 #include <iostream>
3 using namespace std;
4
5 int main ()
6 {
7     for (int n=10; n>0; n--) {
8         if (n==5) continue;
9         cout << n << ", ";
10    }
11    cout << "liftoff!\n";
12 }
```

```
10, 9, 8, 7, 6, 4, 3, 2, 1, liftoff!
```

switch Statement

switch example	if-else equivalent
<pre>switch (x) { case 1: cout << "x is 1"; break; case 2: cout << "x is 2"; break; default: cout << "value of x unknown"; }</pre>	<pre>if (x == 1) { cout << "x is 1"; } else if (x == 2) { cout << "x is 2"; } else { cout << "value of x unknown"; }</pre>

```
1 switch (x) {  
2     case 1:  
3     case 2:  
4     case 3:  
5         cout << "x is 1, 2 or 3";  
6         break;  
7     default:  
8         cout << "x is not 1, 2 nor 3";  
9 }
```


Functions

```
1 // function example
2 #include <iostream>
3 using namespace std;
4
5 int addition (int a, int b)
6 {
7     int r;
8     r=a+b;
9     return r;
10 }
11
12 int main ()
13 {
14     int z;
15     z = addition (5,3);
16     cout << "The result is " << z;
17 }
```

The result is 8

Functions

```
1 // function example
2 #include <iostream>
3 using namespace std;
4
5 int subtraction (int a, int b)
6 {
7     int r;
8     r=a-b;
9     return r;
10 }
11
12 int main ()
13 {
14     int x=5, y=3, z;
15     z = subtraction (7,2);
16     cout << "The first result is " << z << '\n';
17     cout << "The second result is " << subtraction (7,2) << '\n';
18     cout << "The third result is " << subtraction (x,y) << '\n';
19     z= 4 + subtraction (x,y);
20     cout << "The fourth result is " << z << '\n';
21 }
```

```
The first result is 5
The second result is 5
The third result is 2
The fourth result is 6
```

Functions

```
1 // void function example
2 #include <iostream>
3 using namespace std;
4
5 void printmessage ()
6 {
7     cout << "I'm a function!";
8 }
9
10 int main ()
11 {
12     printmessage ();
13 }
```

I'm a function!

Functions

– return value from main

value	description
0	The program was successful
<code>EXIT_SUCCESS</code>	The program was successful (same as above). This value is defined in header <code><cstdlib></code> .
<code>EXIT_FAILURE</code>	The program failed. This value is defined in header <code><cstdlib></code> .

Functions

–pass by value vs. pass by reference

```
1 // passing parameters by reference
2 #include <iostream>
3 using namespace std;
4
5 void duplicate (int& a, int& b, int& c)
6 {
7     a*=2;
8     b*=2;
9     c*=2;
10 }
11
12 int main ()
13 {
14     int x=1, y=3, z=7;
15     duplicate (x, y, z);
16     cout << "x=" << x << ", y=" << y << ", z=" << z;
17     return 0;
18 }
```

x=2, y=6, z=14

Functions

–inline functions

```
1 inline string concatenate (const string& a, const string& b)
2 {
3     return a+b;
4 }
```

Functions

– default values for parameters

```
1 // default values in functions
2 #include <iostream>
3 using namespace std;
4
5 int divide (int a, int b=2)
6 {
7     int r;
8     r=a/b;
9     return (r);
10 }
11
12 int main ()
13 {
14     cout << divide (12) << '\n';
15     cout << divide (20,4) << '\n';
16     return 0;
17 }
```

```
6
5
```

Functions

–function prototypes

```
1 // declaring functions prototypes
2 #include <iostream>
3 using namespace std;
4
5 void odd (int x);
6 void even (int x);
7
8 int main()
9 {
10     int i;
11     do {
12         cout << "Please, enter number (0 to exit): ";
13         cin >> i;
14         odd (i);
15     } while (i!=0);
16     return 0;
17 }
18
19 void odd (int x)
20 {
21     if ((x%2)!=0) cout << "It is odd.\n";
22     else even (x);
23 }
24
25 void even (int x)
26 {
27     if ((x%2)==0) cout << "It is even.\n";
28     else odd (x);
29 }
```

```
Please, enter number (0 to exit): 9
It is odd.
Please, enter number (0 to exit): 6
It is even.
Please, enter number (0 to exit): 1030
It is even.
Please, enter number (0 to exit): 0
It is even.
```


Functions

- recursion

```
1 // factorial calculator
2 #include <iostream>
3 using namespace std;
4
5 long factorial (long a)
6 {
7     if (a > 1)
8         return (a * factorial (a-1));
9     else
10         return 1;
11 }
12
13 int main ()
14 {
15     long number = 9;
16     cout << number << "! = " << factorial (number);
17     return 0;
18 }
```

9! = 362880

Scope

– global vs. local variables

```
1 int foo;           // global variable
2
3 int some_function ()
4 {
5     int bar;       // local variable
6     bar = 0;
7 }
8
9 int other_function ()
10 {
11     foo = 1;  // ok: foo is a global variable
12     bar = 2;  // wrong: bar is not visible from this function
13 }
```

– general rule: DO NOT USE

Scope

- name can only represent one entity

```
1 int some_function ()
2 {
3     int x;
4     x = 0;
5     double x;    // wrong: name already used in this scope
6     x = 0.0;
7 }
```

Scope

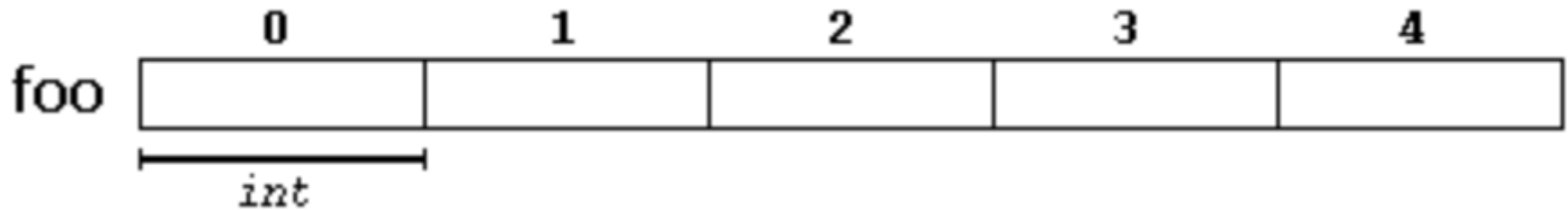
- block scope

```
1 // inner block scopes
2 #include <iostream>
3 using namespace std;
4
5 int main () {
6     int x = 10;
7     int y = 20;
8     {
9         int x;    // ok, inner scope.
10        x = 50;    // sets value to inner x
11        y = 50;    // sets value to (outer) y
12        cout << "inner block:\n";
13        cout << "x: " << x << '\n';
14        cout << "y: " << y << '\n';
15    }
16    cout << "outer block:\n";
17    cout << "x: " << x << '\n';
18    cout << "y: " << y << '\n';
19    return 0;
20 }
```

```
inner block:
x: 50
y: 50
outer block:
x: 10
y: 50
```

Arrays

- contiguous memory locations

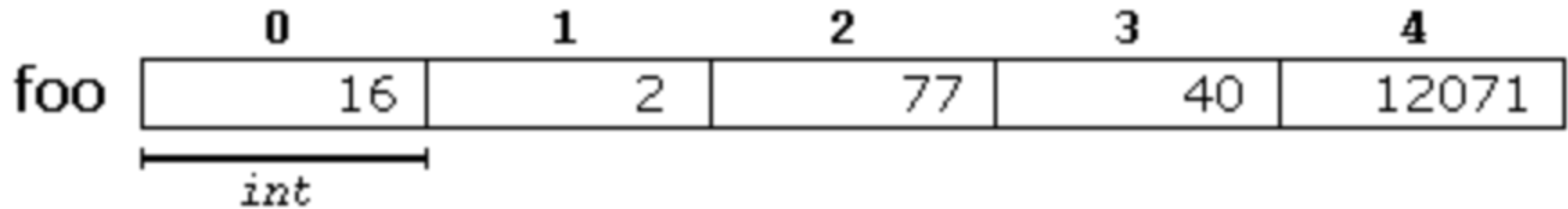


```
int foo [5];
```

Initializing Arrays

- elements not automatically initialized, but can be explicitly initialized

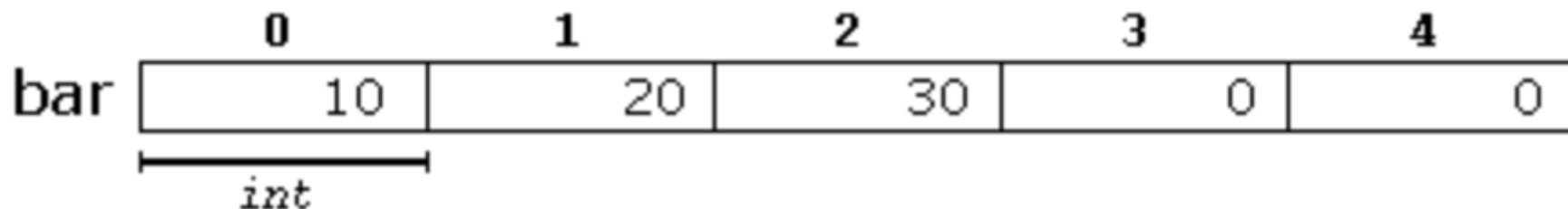
```
int foo [5] = { 16, 2, 77, 40, 12071 };
```



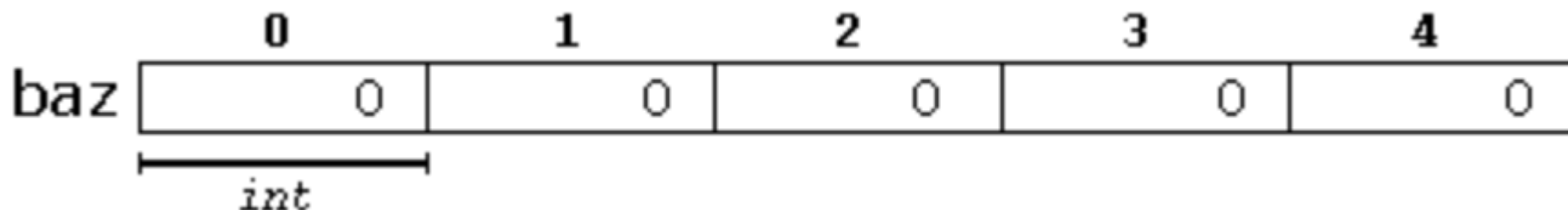
Initializing Arrays

- if { } are present, values are initialized to default values

```
int bar [5] = { 10, 20, 30 };
```



```
int baz [5] = { };
```



Initializing Arrays

- initialized arrays without size are automatically sized to accommodate values

```
int foo [] = { 16, 2, 77, 40, 12071 };
```

- can be initialized without \equiv

```
1 int foo[] = { 10, 20, 30 };  
2 int foo[] { 10, 20, 30 };
```

- no error if range exceeded
- example uses of arrays

```
1 foo[0] = a;  
2 foo[a] = 75;  
3 b = foo [a+2];  
4 foo[foo[a]] = foo[2] + 5;
```


Arrays

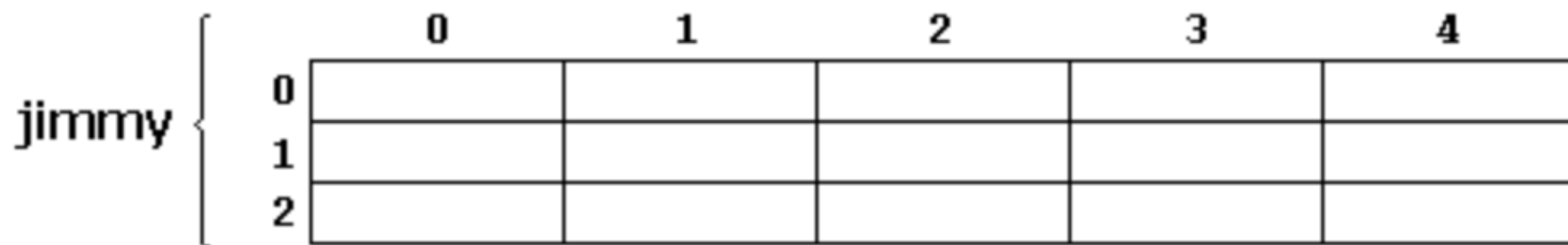
```
1 // arrays example
2 #include <iostream>
3 using namespace std;
4
5 int foo [] = {16, 2, 77, 40, 12071};
6 int n, result=0;
7
8 int main ()
9 {
10     for ( n=0 ; n<5 ; ++n )
11     {
12         result += foo[n];
13     }
14     cout << result;
15     return 0;
16 }
```

12206

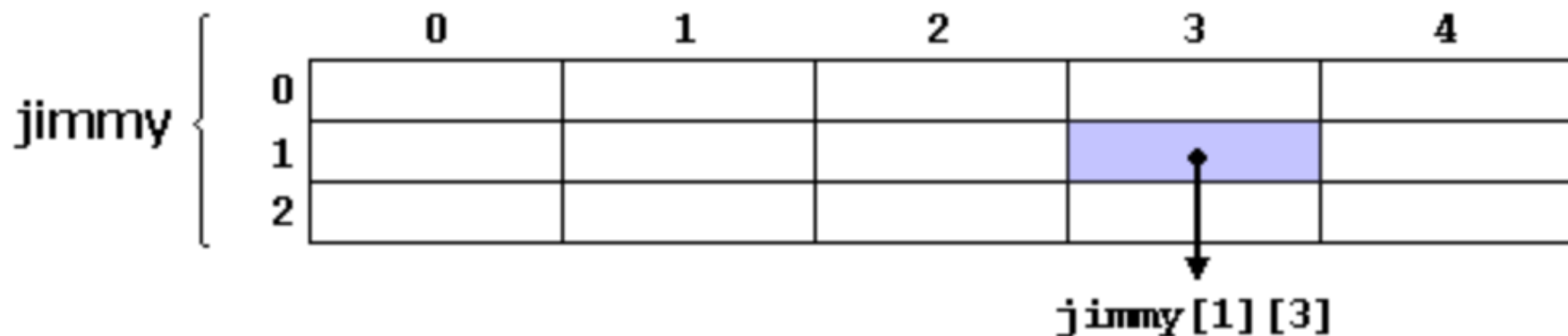
Multidimensional Arrays

– arrays of arrays

```
int jimmy [3][5];
```



```
jimmy[1][3]
```



Multidimensional Arrays

- can be any dimension, but space increases exponentially

```
char century [100][365][24][60][60];
```

- allocates a char for each second in the last century
 - consumes 3GB of memory
-
- could have been implemented as a single-dimension array

```
1 int jimmy [3][5];    // is equivalent to
2 int jimmy [15];      // (3 * 5 = 15)
```

Multidimensional Arrays

multidimensional array	pseudo-multidimensional array
<pre>#define WIDTH 5 #define HEIGHT 3 int jimmy [HEIGHT][WIDTH]; int n,m; int main () { for (n=0; n<HEIGHT; n++) for (m=0; m<WIDTH; m++) { jimmy[n][m]=(n+1)*(m+1); } }</pre>	<pre>#define WIDTH 5 #define HEIGHT 3 int jimmy [HEIGHT * WIDTH]; int n,m; int main () { for (n=0; n<HEIGHT; n++) for (m=0; m<WIDTH; m++) { jimmy[n*WIDTH+m]=(n+1)*(m+1); } }</pre>

		0	1	2	3	4
jimmy {	0	1	2	3	4	5
	1	2	4	6	8	10
	2	3	6	9	12	15

Arrays as Parameters

```
1 // arrays as parameters
2 #include <iostream>
3 using namespace std;
4
5 void printarray (int arg[], int length) {
6     for (int n=0; n<length; ++n)
7         cout << arg[n] << ' ';
8     cout << '\n';
9 }
10
11 int main ()
12 {
13     int firstarray[] = {5, 10, 15};
14     int secondarray[] = {2, 4, 6, 8, 10};
15     printarray (firstarray,3);
16     printarray (secondarray,5);
17 }
```

5 10 15

2 4 6 8 10

```
void procedure (int myarray[][3][4])
```

Character Arrays

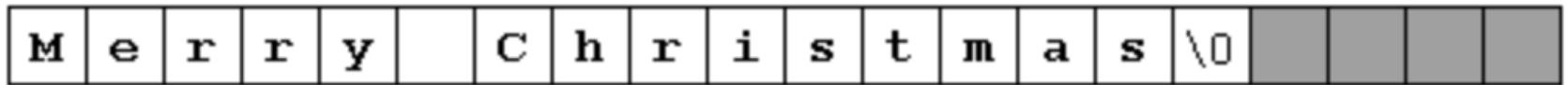
- sets aside space, but not initialized

```
char foo [20];
```

foo



foo



Character Arrays

- can initialize with individual elements or string literals

```
1 char myword[] = { 'H', 'e', 'l', 'l', 'o', '\0' };  
2 char myword[] = "Hello";
```

- not valid

```
1 myword = "Bye";  
2 myword[] = "Bye";
```

```
myword = { 'B', 'y', 'e', '\0' };
```

- OK

```
1 myword[0] = 'B';  
2 myword[1] = 'y';  
3 myword[2] = 'e';  
4 myword[3] = '\0';
```

Strings and Character Arrays

```
1 // strings and NTCS:
2 #include <iostream>
3 #include <string>
4 using namespace std;
5
6 int main ()
7 {
8     char question1[] = "What is your name? ";
9     string question2 = "Where do you live? ";
10    char answer1 [80];
11    string answer2;
12    cout << question1;
13    cin >> answer1;
14    cout << question2;
15    cin >> answer2;
16    cout << "Hello, " << answer1;
17    cout << " from " << answer2 << "!\n";
18    return 0;
19 }
```

```
What is your name? Homer
Where do you live? Greece
Hello, Homer from Greece!
```


Strings and Character Arrays

– can be transformed one to another

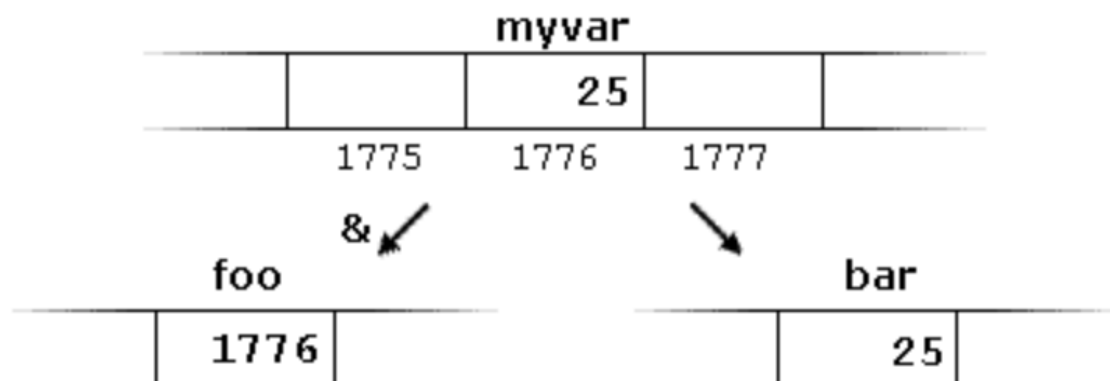
```
1 char myntcs[] = "some text";  
2 string mystring = myntcs;    // convert c-string to string  
3 cout << mystring;           // printed as a library string  
4 cout << mystring.c_str();   // printed as a c-string
```

Pointers

- pointer – the address of something
- exact memory locations unknown at compile time
- use & to get the address of a variable

```
foo = &myvar;
```

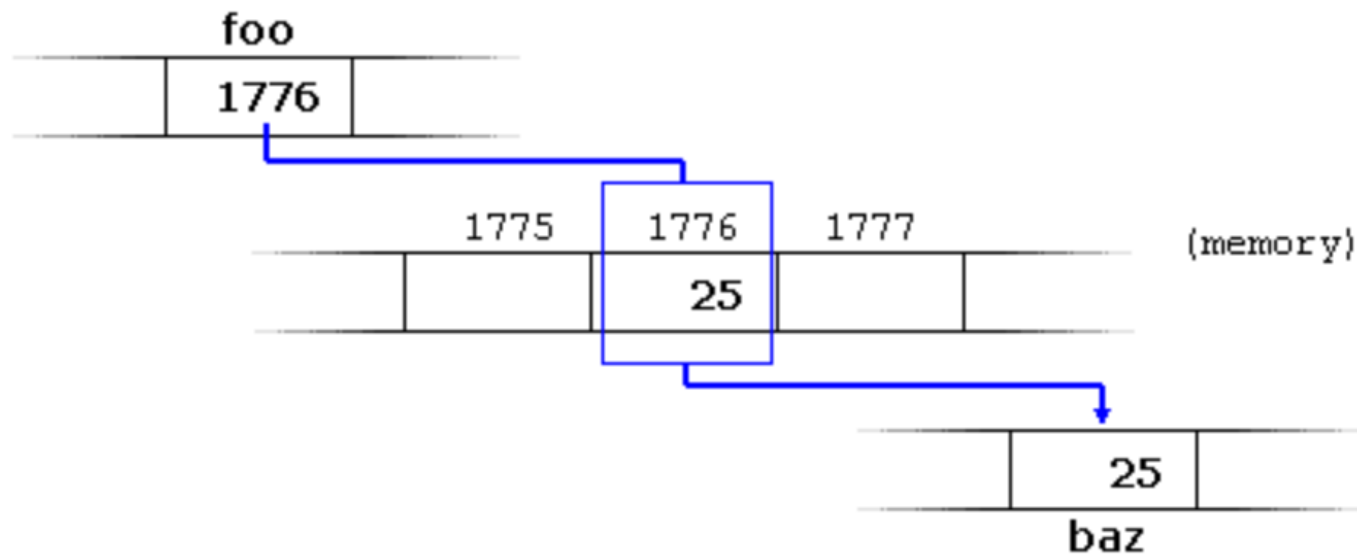
```
1 myvar = 25;  
2 foo = &myvar;  
3 bar = myvar;
```



Pointers

- use * to get the value at a pointer (address)

```
baz = *foo;
```



Pointers

– & and * are complementary

```
1 baz = foo;    // baz equal to foo (1776)
2 baz = *foo;   // baz equal to value pointed to by foo (25)
```

– with following assignments

```
1 myvar = 25;
2 foo = &myvar;
```

– all of the following are true

```
1 myvar == 25
2 &myvar == 1776
3 foo == 1776
4 *foo == 25
```

```
*foo == myvar
```

Declaring Pointers

- all are the same size in memory

```
1 int * number;  
2 char * character;  
3 double * decimals;
```

- different

```
int * p1, * p2;
```

```
int * p1, p2;
```

Pointers

```
1 // my first pointer
2 #include <iostream>
3 using namespace std;
4
5 int main ()
6 {
7     int firstvalue, secondvalue;
8     int * mypointer;
9
10    mypointer = &firstvalue;
11    *mypointer = 10;
12    mypointer = &secondvalue;
13    *mypointer = 20;
14    cout << "firstvalue is " << firstvalue << '\n';
15    cout << "secondvalue is " << secondvalue << '\n';
16    return 0;
17 }
```

```
firstvalue is 10
secondvalue is 20
```

Pointers

```
1 // more pointers
2 #include <iostream>
3 using namespace std;
4
5 int main ()
6 {
7     int firstvalue = 5, secondvalue = 15;
8     int * p1, * p2;
9
10    p1 = &firstvalue; // p1 = address of firstvalue
11    p2 = &secondvalue; // p2 = address of secondvalue
12    *p1 = 10;          // value pointed to by p1 = 10
13    *p2 = *p1;         // value pointed to by p2 = value pointed to by p1
14    p1 = p2;           // p1 = p2 (value of pointer is copied)
15    *p1 = 20;          // value pointed to by p1 = 20
16
17    cout << "firstvalue is " << firstvalue << '\n';
18    cout << "secondvalue is " << secondvalue << '\n';
19    return 0;
20 }
```

```
firstvalue is 10
secondvalue is 20
```

Pointers and Arrays

- array name with no index is a pointer to the first element
- arrays can always be converted to pointers

```
1 int myarray [20];  
2 int * mypointer;
```

```
mypointer = myarray;
```

- not valid to go the other way

```
myarray = mypointer;
```


Pointers and Arrays

```
1 // more pointers
2 #include <iostream>
3 using namespace std;
4
5 int main ()
6 {
7     int numbers[5];
8     int * p;
9     p = numbers;  *p = 10;
10    p++;  *p = 20;
11    p = &numbers[2];  *p = 30;
12    p = numbers + 3;  *p = 40;
13    p = numbers;  *(p+4) = 50;
14    for (int n=0; n<5; n++)
15        cout << numbers[n] << ", ";
16    return 0;
17 }
```

10, 20, 30, 40, 50,

Pointers and Arrays

- array with index is simply a pointer with an offset
- can be represented with pointer

```
1 a[5] = 0;           // a [offset of 5] = 0
2 *(a+5) = 0;         // pointed to by (a+5) = 0
```

Pointer Initialization

- pointers can be initialized at declaration

```
1 int myvar;  
2 int * myptr = &myvar;
```

- same as

```
1 int myvar;  
2 int * myptr;  
3 myptr = &myvar;
```

- not valid

```
1 int myvar;  
2 int * myptr;  
3 *myptr = &myvar;
```

- OK

```
1 int myvar;  
2 int *foo = &myvar;  
3 int *bar = foo;
```

Pointer Arithmetic

- pointers can be used in arithmetic expressions, with underlying size taken into account
- suppose the following have addresses 1000, 2000, 3000

```
1 char *mychar;  
2 short *myshort;  
3 long *mylong;
```

- after the following

```
1 ++mychar;  
2 ++myshort;  
3 ++mylong;
```

- values are 1001, 2002, 3004
- same results for

```
1 mychar = mychar + 1;  
2 myshort = myshort + 1;  
3 mylong = mylong + 1;
```

Pointer Arithmetic

– the following is equivalent to *(p++)

```
*p++
```

– other examples

```
1 *p++    // same as *(p++): increment pointer, and dereference unincremented address
2 ++*p    // same as *(++p): increment pointer, and dereference incremented address
3 ++*p    // same as ++(*p): dereference pointer, and increment the value it points to
4 (*p)++  // dereference pointer, and post-increment the value it points to
```

– assignment done before increment

```
*p++ = *q++;
```

– same as

```
1 *p = *q;
2 ++p;
3 ++q;
```

Pointers and const

- if value pointed to is const, it cannot be modified

```
1 int x;  
2 int y = 10;  
3 const int * p = &y;  
4 x = *p;           // ok: reading p  
5 *p = x;           // error: modifying p, which is const-qualified
```

- pointers can be const

```
1 int x;  
2     int *      p1 = &x;  // non-const pointer to non-const int  
3 const int *    p2 = &x;  // non-const pointer to const int  
4     int * const p3 = &x;  // const pointer to non-const int  
5 const int * const p4 = &x; // const pointer to const int
```

- same

```
1 const int * p2a = &x;  // non-const pointer to const int  
2 int const * p2b = &x;  // also non-const pointer to const int
```

Pointers and const

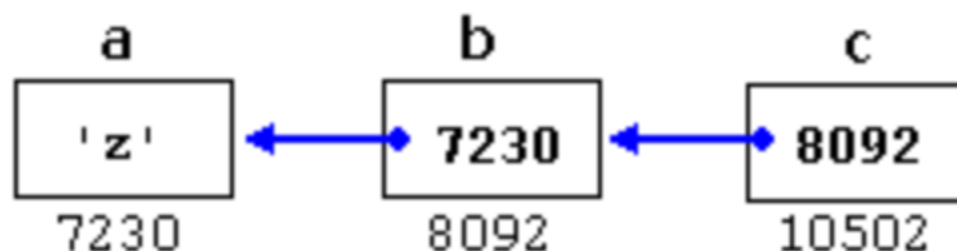
– pointers are not const, so can be modified

```
1 // pointers as arguments:
2 #include <iostream>
3 using namespace std;
4
5 void increment_all (int* start, int* stop)
6 {
7     int * current = start;
8     while (current != stop) {
9         ++(*current); // increment value pointed
10        ++current;     // increment pointer
11    }
12 }
13
14 void print_all (const int* start, const int* stop)
15 {
16     const int * current = start;
17     while (current != stop) {
18         cout << *current << '\n';
19         ++current; // increment pointer
20     }
21 }
22
23 int main ()
24 {
25     int numbers[] = {10,20,30};
26     increment_all (numbers,numbers+3);
27     print_all (numbers,numbers+3);
28     return 0;
29 }
```

```
11
21
31
```

Pointers to Pointers

```
1 char a;  
2 char * b;  
3 char ** c;  
4 a = 'z';  
5 b = &a;  
6 c = &b;
```



void Pointers

– void pointers point to no particular type

```
1 // increaser
2 #include <iostream>
3 using namespace std;
4
5 void increase (void* data, int psize)
6 {
7     if ( psize == sizeof(char) )
8     { char* pchar; pchar=(char*)data; ++(*pchar); }
9     else if (psize == sizeof(int) )
10    { int* pint; pint=(int*)data; ++(*pint); }
11 }
12
13 int main ()
14 {
15     char a = 'x';
16     int b = 1602;
17     increase (&a, sizeof(a));
18     increase (&b, sizeof(b));
19     cout << a << ", " << b << '\n';
20     return 0;
21 }
```

y, 1603

Pointers

- pointers can point to any address

```
1 int * p;           // uninitialized pointer (local variable)
2
3 int myarray[10];
4 int * q = myarray+20; // element out of bounds
```

- pointers can point to nothing

```
1 int * p = 0;
2 int * q = nullptr;
```

- or simply

```
int * r = NULL;
```

- NULL pointers and void pointers are different

Dynamic Memory

- memory can be allocated during run time with new

```
1 int * foo;  
2 foo = new int [5];
```

- can check for success/failure

```
1 int * foo;  
2 foo = new (nothrow) int [5];  
3 if (foo == nullptr) {  
4     // error assigning memory. Take measures.  
5 }
```

Dynamic Memory

- memory can (and should) be de-allocated during run time with delete

```
1 delete pointer;  
2 delete[] pointer;
```

- can also use malloc/free (from C), but don't mix

Dynamic Memory

```
1 // rememb-o-matic
2 #include <iostream>
3 #include <new>
4 using namespace std;
5
6 int main ()
7 {
8     int i,n;
9     int * p;
10    cout << "How many numbers would you like to type? ";
11    cin >> i;
12    p= new (nothrow) int[i];
13    if (p == nullptr)
14        cout << "Error: memory could not be allocated";
15    else
16    {
17        for (n=0; n<i; n++)
18        {
19            cout << "Enter number: ";
20            cin >> p[n];
21        }
22        cout << "You have entered: ";
23        for (n=0; n<i; n++)
24            cout << p[n] << ", ";
25        delete[] p;
26    }
27    return 0;
28 }
```

```
How many numbers would you like to type? 5
Enter number : 75
Enter number : 436
Enter number : 1067
Enter number : 8
Enter number : 32
You have entered: 75, 436, 1067, 8, 32,
```

Data Structures

– struct

```
1 struct product {  
2     int weight;  
3     double price;  
4 } ;  
5  
6 product apple;  
7 product banana, melon;
```

– or

```
1 struct product {  
2     int weight;  
3     double price;  
4 } apple, banana, melon;
```

– access

```
1 apple.weight  
2 apple.price  
3 banana.weight  
4 banana.price  
5 melon.weight  
6 melon.price
```

Data Structures

```
1 // example about structures
2 #include <iostream>
3 #include <string>
4 #include <sstream>
5 using namespace std;
6
7 struct movies_t {
8     string title;
9     int year;
10 } mine, yours;
11
12 void printmovie (movies_t movie);
13
14 int main ()
15 {
16     string mystr;
17
18     mine.title = "2001 A Space Odyssey";
19     mine.year = 1968;
20
21     cout << "Enter title: ";
22     getline (cin,yours.title);
23     cout << "Enter year: ";
24     getline (cin,mystr);
25     stringstream(mystr) >> yours.year;
26
27     cout << "My favorite movie is:\n ";
28     printmovie (mine);
29     cout << "And yours is:\n ";
30     printmovie (yours);
31     return 0;
32 }
33
34 void printmovie (movies_t movie)
35 {
36     cout << movie.title;
37     cout << " (" << movie.year << ")\n";
38 }
```

```
Enter title: Alien
Enter year: 1979
```

```
My favorite movie is:
  2001 A Space Odyssey (1968)
And yours is:
  Alien (1979)
```

Data Structures

```
1 // array of structures
2 #include <iostream>
3 #include <string>
4 #include <sstream>
5 using namespace std;
6
7 struct movies_t {
8     string title;
9     int year;
10 } films [3];
11
12 void printmovie (movies_t movie);
13
14 int main ()
15 {
16     string mystr;
17     int n;
18
19     for (n=0; n<3; n++)
20     {
21         cout << "Enter title: ";
22         getline (cin,films[n].title);
23         cout << "Enter year: ";
24         getline (cin,mystr);
25         stringstream(mystr) >> films[n].year;
26     }
27
28     cout << "\nYou have entered these movies:\n";
29     for (n=0; n<3; n++)
30         printmovie (films[n]);
31     return 0;
32 }
33
34 void printmovie (movies_t movie)
35 {
36     cout << movie.title;
37     cout << " (" << movie.year << ") \n";
38 }
```

```
Enter title: Blade Runner
Enter year: 1982
Enter title: The Matrix
Enter year: 1999
Enter title: Taxi Driver
Enter year: 1976
```

```
You have entered these movies:
Blade Runner (1982)
The Matrix (1999)
Taxi Driver (1976)
```


Data Structures

```
1 // pointers to structures
2 #include <iostream>
3 #include <string>
4 #include <sstream>
5 using namespace std;
6
7 struct movies_t {
8     string title;
9     int year;
10 };
11
12 int main ()
13 {
14     string mystr;
15
16     movies_t amovie;
17     movies_t * pmovie;
18     pmovie = &amovie;
19
20     cout << "Enter title: ";
21     getline (cin, pmovie->title);
22     cout << "Enter year: ";
23     getline (cin, mystr);
24     (stringstream) mystr >> pmovie->year;
25
26     cout << "\nYou have entered:\n";
27     cout << pmovie->title;
28     cout << " (" << pmovie->year << ")\n";
29
30     return 0;
31 }
```

```
Enter title: Invasion of the body snatchers
Enter year: 1978
```

```
You have entered:
Invasion of the body snatchers (1978)
```

Data Structures

– pointers to structs

```
pmovie->title
```

```
(*pmovie).title
```

– different from

```
*pmovie.title
```

```
*(pmovie.title)
```

Expression	What is evaluated	Equivalent
<code>a.b</code>	Member <code>b</code> of object <code>a</code>	
<code>a->b</code>	Member <code>b</code> of object pointed to by <code>a</code>	<code>(*a).b</code>
<code>*a.b</code>	Value pointed to by member <code>b</code> of object <code>a</code>	<code>*(a.b)</code>

Data Structures

- nested structs

```
1 struct movies_t {  
2     string title;  
3     int year;  
4 };  
5  
6 struct friends_t {  
7     string name;  
8     string email;  
9     movies_t favorite_movie;  
10 } charlie, maria;  
11  
12 friends_t * pfriends = &charlie;
```

- access

```
1 charlie.name  
2 maria.favorite_movie.title  
3 charlie.favorite_movie.year  
4 pfriends->favorite_movie.year
```

Other Data Structures

–type aliases

```
1 typedef char C;  
2 typedef unsigned int WORD;  
3 typedef char * pChar;  
4 typedef char field [50];
```

–can be used as

```
1 C mychar, anotherchar, *ptc1;  
2 WORD myword;  
3 pChar ptc2;  
4 field name;
```

–with using clause

```
1 using C = char;  
2 using WORD = unsigned int;  
3 using pChar = char *;  
4 using field = char [50];
```

Other Data Structures

– union

```
1 union mytypes_t {  
2     char c;  
3     int i;  
4     float f;  
5 } mytypes;
```

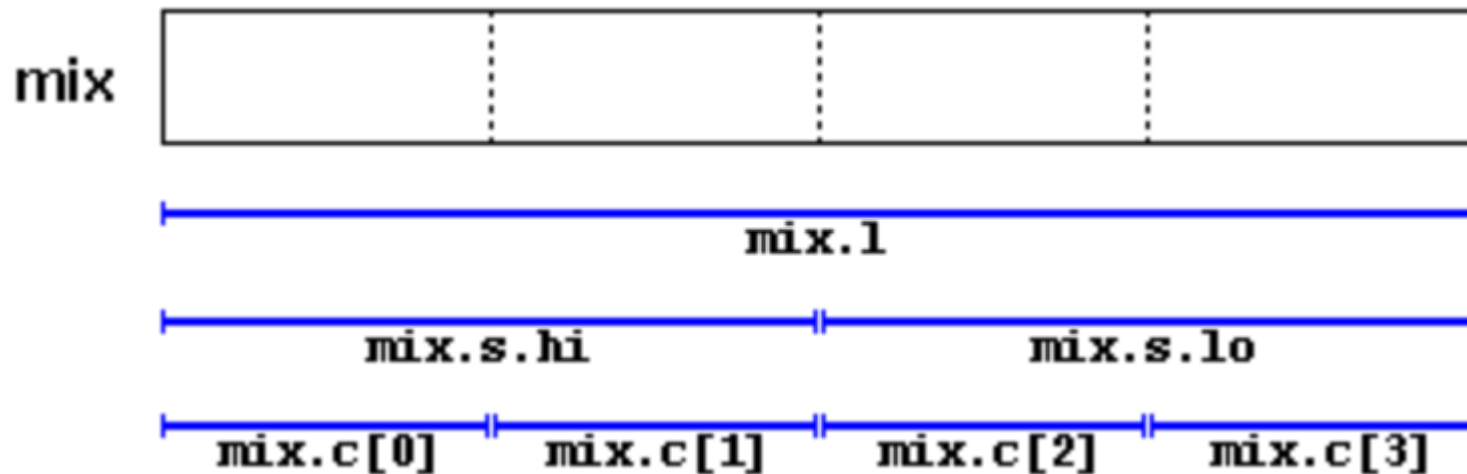
– can be accessed as

```
1 mytypes.c  
2 mytypes.i  
3 mytypes.f
```

Other Data Structures

– union

```
1 union mix_t {  
2     int l;  
3     struct {  
4         short hi;  
5         short lo;  
6     } s;  
7     char c[4];  
8 } mix;
```



Other Data Structures

– anonymous union

structure with regular union	structure with anonymous union
<pre>struct book1_t { char title[50]; char author[50]; union { float dollars; int yen; } price; } book1;</pre>	<pre>struct book2_t { char title[50]; char author[50]; union { float dollars; int yen; }; } book2;</pre>

```
1 book1.price.dollars  
2 book1.price.yen
```

```
1 book2.dollars  
2 book2.yen
```

Enumerated Types

```
enum colors_t {black, blue, green, cyan, red, purple, yellow, white};
```

```
1 colors_t mycolor;  
2  
3 mycolor = blue;  
4 if (mycolor == green) mycolor = red;
```

– can assign integer values (assigned anyway starting at 0)

```
1 enum months_t { january=1, february, march, april,  
2               may, june, july, august,  
3               september, october, november, december} y2k;
```