

Programming Constructs

Topics

- Program structure
- Declarations
- Operators
- Control Flow
 - If-then-else
 - Varieties of Loops
 - Switch Statements

C++ Program Structure

includes of header files, similar to importing packages in Java

```
int           // return type
main()        // entry point in code
{             // beginning of main
    declaration // define names & types of variables
    statements  // code
    return 0;   // return value of your program
}              // end of main
```

Example Program

```
#include <iostream>

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

Equivalent Example Program

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

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

Breaking down the code

`#include <iostream>`

- Include the contents of the file iostream.h
 - Case sensitive – lower case only
- No semicolon at the end of line

`int main() or main(int argc, char* argv[])`

- This function is called when the program starts running.

`cout`

- Prints out a string or series of strings

C++ Preprocessor

```
#define CONSTANT1 "You will learn a lot \
in this class"
```

```
int main(int argc, char* argv[])
{
    cout << CONSTANT1;
    return 0;
}
```

After the preprocessor

```
int main(int argc, char* argv)
{
    cout << "You will learn a lot in this class";
    return 0;
}
```

Conditional Compilation

```
#define ECE106

int main(int argc, char* argv)
{
    #ifdef ECE106
        cout << "You will learn a lot in this class";
    #else
        cout << "Some other class";
    #endif
    return 0;
}
```

Like Java, like C

Operators same as Java:

- Arithmetic
 - `i = i+1; i++; i--; i *= 2;`
 - `+, -, *, /, %,`
- Relational and Logical
 - `<, >, <=, >=, ==, !=`
 - `&&, ||, &, |, !`

Syntax same as in Java:

- `if ( ) { } else { }`
- `while ( ) { }`
- `do { } while ( );`
- `for(i=1; i <= 100; i++) { }`
- `switch ( ) {case 1: ... }`
- `continue; break;`

Precedence of C++ operators

Operators	Associativity
<code>() [] -> .</code>	left to right
<code>! ~ ++ -- + - * & (type) sizeof</code>	right to left
<code>* / %</code>	left to right
<code>+ -</code>	left to right
<code><< >></code>	left to right
<code>< <= > >=</code>	left to right
<code>== !=</code>	left to right
<code>&</code>	left to right
<code>^</code>	left to right
<code> </code>	left to right
<code>&&</code>	left to right
<code> </code>	left to right
<code>?:</code>	right to left
<code>= += -= *= /= %= &= ^= != <<= >>=</code>	right to left
<code>,</code>	left to right

Note: Unary `+`, `-`, and `*` have higher precedence than binary forms

Simple Data Types

datatype	size	values
char	1	-128 to 127
short	2	-32,768 to 32,767
int	4	-2,147,483,648 to 2,147,483,647
long	4	-2,147,483,648 to 2,147,483,647
float	4	3.4E+/-38 (7 digits)
double	8	1.7E+/-308 (15 digits long)

Can do type casts just like in Java

Can find out the size of any type or object with sizeof (type)

Programmer gotchas

Uninitialized variables

- catch with `-Wall` compiler option

```
#include <stdio.h>
```

```
int main(int argc, char* argv[])
{
    int i;
    factorial(i);
    return 0;
}
```

If Example

C++ Code

```
int max(int x, int y)
{
    if (x > y)
        return x;
    else
        return y;
}
```

“Do-While” Loop Example

C++ Code

```
int func
(int x)
{
    int result = 1;
    do {
        result *= x;
        x = x-1;
    } while (x > 1);
    return result;
}
```

- Only continue looping when “while” condition holds

General “Do-While” Translation

C++ Code

```
do  
    Body  
while (Test);
```

- *Body* can be any C statement
 - Typically compound statement:

```
{  
    Statement1;  
    Statement2;  
    ...  
    Statementn;  
}
```

- *Test* is expression returning integer
 - = 0 interpreted as false ≠0 interpreted as true

“While” Loop Example

C++ Code

```
int func_while  
(int x)  
{  
    int result = 1;  
    while (x > 1) {  
        result *= x;  
        x = x-1;  
    };  
    return result;  
}
```

- Is this code equivalent to the do-while version?

“For” Loop Example

```
int func_for(int x, unsigned p) {  
    int result;  
    for (result = 1, int i = 0; i < p; i++) {  
        result *= x;  
    }  
    return result;  
}
```

“For” Loop Example

```
for (result = 0,  
     int i = 1; i < p;  
     i++) {  
    result *= x;  
}
```

General Form

```
for (Init; Test; Update )  
    Body
```

Init

```
result =  
0, i = 1
```

Test

```
i < p
```

Update

```
i++
```

Body

```
{  
    result *= x;  
}
```

“For” → “While”

For Version

```
for (Init; Test; Update )  
    Body
```

While Version

```
Init;  
while (Test) {  
    Body  
    Update ;  
}
```

Do-While Version

```
Init;  
if (Test)  
do {  
    Body  
    Update ;  
} while (Test);
```

Switch Statements

Implementation Options

- Series of conditionals
 - Avoids many if's
 - Possible when cases are small integer constants
- In example code
 - No default given
 - Instead of return can have break

```
typedef enum  
{ADD, MULT, MINUS, DIV, MOD, BAD}  
op_type;  
  
char unparse_symbol(op_type op)  
{  
    switch (op) {  
        case ADD :  
            return '+';  
        case MULT:  
            return '*';  
        case MINUS:  
            return '-';  
        case DIV:  
            return '/';  
        case MOD:  
            return '%';  
        case BAD:  
            return '?';  
    }  
}
```

```

typedef enum
{ADD, MULT, MINUS, DIV, MOD, BAD}
  op_type;

char unparse_symbol(op_type op)
{
    switch (op) {
        case ADD :
            return '+';
        case MULT:
            return '*';
        case MINUS:
            return '-';
        case DIV:
            return '/';
        case MOD:
            return '%';
        case BAD:
            return '?';
        default: return '0';//no match
    }
}

```

Switch Statements

Implementation Options

- Series of conditionals
 - Avoids many if's
 - Possible when cases are small integer constants
- In example code
 - No default given
 - Instead of return can have break

```

typedef enum
{ADD, MULT, MINUS, DIV, MOD, BAD}
  op_type;

char unparse_symbol(op_type op){
    int val;
    switch (op) {
        case ADD :
            val = '+'; break;
        case MULT:
            val = '*'; break;
        case MINUS:
            val = '-'; break;
        case DIV:
            val = '/'; break;
        case MOD:
            val = '%'; break;
        case BAD:
            val = '?'; break;
        default: val = '0';//no match
    }
    return val;
}

```

Switch Statements

Implementation Options

- Series of conditionals
 - Avoids many if's
 - Possible when cases are small integer constants
- In example code
 - No default given
 - Instead of return can have break

```

typedef enum
{ADD, MULT, MINUS, DIV, MOD, BAD}
  op_type;

char unparse_symbol(op_type op){
  int val;
  switch (op) {
    case ADD :
      val = '+';
    case MULT:
      val = '*';
    case MINUS:
      val = '-';
    case DIV:
      val = '/';
    case MOD:
      val = '%';
    case BAD:
      val = '?';
    default: val = '0';
  }
  return val;
}

```

Switch Statements

Implementation Options

- Series of conditionals
 - Avoids many if's
 - Possible when cases are small integer constants
- In example code
 - What happens if I omit the breaks ??
 - Fall-through after the first match
 - In the example, the returned value will always be '0'

Summarizing

C Control

- if-then-else
- do-while
- while
- switch

Memory Organization: Bytes

Topics

- Why bits?
- Representing information as bits
 - Binary/Hexadecimal
 - Byte representations
 - » numbers
 - » characters and strings
 - » Instructions

Binary Representations

Base 2 Number Representation

- Represent 15213_{10} as 11101101101101_2
- Represent 1.20_{10} as $1.0011001100110011[0011]..._2$
- 0/1 are bits
- A byte = 8 bits

Byte-Oriented Memory Organization

Programs Refer to Memory Addresses

- Conceptually very large array of bytes
- Address space private to particular “process”
 - Program being executed
 - Program can clobber its own data, but not that of others

Compiler + Run-Time System Control Allocation

- Where different program objects should be stored
- Multiple mechanisms: static, stack, and heap
- In any case, all allocation within single address space

Encoding Byte Values

Byte = 8 bits

- Binary 00000000_2 to 11111111_2
- Decimal: 0_{10} to 255_{10}
- Hexadecimal 00_{16} to FF_{16}
 - Base 16 number representation
 - Use characters '0' to '9' and 'A' to 'F'
 - Write $FA1D37B_{16}$ in C as `0xFA1D37B`
» Or `0xfa1d37b`

	Hex	Decimal	Binary
0	0	0000	
1	1	0001	
2	2	0010	
3	3	0011	
4	4	0100	
5	5	0101	
6	6	0110	
7	7	0111	
8	8	1000	
9	9	1001	
A	10	1010	
B	11	1011	
C	12	1100	
D	13	1101	
E	14	1110	
F	15	1111	

Machine Words

Machine Has “Word Size”

- Nominal size of integer-valued data
 - Including addresses
- Machines support multiple data formats
 - Fractions or multiples of word size
 - Always integral number of bytes

Main Points

It's All About Bits & Bytes

- Numbers
- Programs
- Data has size in bytes
- Word size