C++/Java

Java: a “pure” object-oriented language.

C++: a “hybrid” language. Allows multiple programming styles.

Kernel C++/ a better C: the subset of C++ without OO-features.

C++: Hello World

```
//Hello world in C++

#include <iostream>    //IO library
#include <string>      //string type

using namespace std;

void pr_message(string s = “Hello world!”)
  { cout << s << endl; }

int main()
{ pr_message();
  pr_message(“Hello C++/Java programmers!”);
}
```

A Kernel C++ program.
No classes/objects

Java : Hello World

```
// Hello world in Java

public class hello{
  static void pr_message(String s)
    {System.out.println(s);}

  public static void main(String[] args)
    { pr_message(“Hello world!”);
      pr_message(“Hello C++/Java programmers!”);
    }

}
```

Every thing is a
Class/object

Statements

C++ statements: almost like Java Statements: p.375, BS p.132

Expression Statement

int j;

- ++ j;
- j= j + 2;
- j+= 2;
- cout << “Hello\n”;
- j + 2 ;

expression ;

Compound Statement

{tmp=a; a= b; b= tmp;}

{ *statement statement ... statement* }

Declaration Statement

A declaration is a statement (we will consider declarations later).

{ **int** i=0; i= i+1; **double** b; cin >> b; …}

if/if-else Statements

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

void check_temp(int temperature)
{  if (temperature >= 32)
          cout << “Above Freezing!\n”;
   cout << “Fahrenheit is “ << temperature << endl;
}

void check_grade (int grade)
{  char letter_gr = ‘F’;
   if (grade > 70 && grade < 80)
        {  cout << “ you passed “;
           letter_gr = ‘C’;
        }
   cout << “ Letter grade is “ << letter_gr << endl;
}

void get_min(int x, int y)
{  int min = 0;
   if (x < y) min = x;  else  min = y;
   cout << “min = “ << min << endl
}
```

if (*condition*) *statement*

if (*condition*) *statement*

if (*condition*) *statement1*
else *statement2*

```
int main()
{  check_temp(2); check_temp(32); check_temp(45);
   check_grade(10); check_grade(75);
   get_min(10,2); get_min(2, 10001);
}
```

```
Fahrenheit is 2
Above Freezing!
Fahrenheit is 32
Above Freezing!
Fahrenheit is 45
 Letter grade is F
 you passed  Letter grade is C
min = 2
min = 2
```


while/for/do statements

while:

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

int main()
{  int i = 0, sum = 0;
   while (i <= 10) {
        sum += i;  ++i;
        }
   cout << “\nSum for i “<< i << “ is “ << sum << endl;
}
```

while (*condition*)
statement1

do:

```
   int n = 0;
   do {  cout << “\nEnter a positive integer: “;
         cin >> n;
      } while (n <= 0);
```

do *statement*
while (*condition*)

for:

for (*for-init-statement condition*; *expression*)
statement

```
int i = 0, sum = 0;
for (i = 1; i <= 10; ++i) sum += i;
cout << “Sum= “ << sum << endl;
.....
int sum = 0; cout << “Enter five numbers:\n”
for (int i = 1; i <= 5; ++i)
   {int tmp; cin >> tmp; sum += tmp;}
cout << “Sum= “ << sum << endl;
```

break/continue/goto statements

```
double x;

for (int i = 0; i < 10; ++i) {
  cout << "\nEnter a positive number for square root: ";
  cin >> x;
  if (x < 0.0){  cout << "All done" << endl;
                 break;    //exit loop if value is negative
              }
  cout << sqrt(x) << endl;
}
cout << "\n Number of square roots " << i << endl;
```

Illegal:
i isn't known here!

```
char c;

for (int i = 0; i < MAX; ++i) {
    cin.get(c);
    if (isdigit(c)) continue;
    /*proces non-digits*/ .....

    //continue jumps to here
}
```

break/continue interrupts nearest surrounding iterative statement.

Java has labeled break, C++ has goto

Java:

```
int A[100][100];

searchForKey:  // a Java search of a double nested loop
for (int i= 0; i< 100; i++)
  for (int j=0; j<100; ++j)
    if (A[i][j] == Key) break searchForKey;
```

not in C++

C++ :

goto-statements often used for breaking out of multiple levels of looping:

goto *label;*

```
for(int i; i<100; ++i)
  for(int j=0; j<100; ++j)
    if(A[i][j]== Key) goto FoundKey;

FoundKey: //execution comes here on found key
```

goto not in Java !!

Use of **goto** usually considered bad programming.

switch statement

```
int a_grades = 0, b_grades = 0, c_grades = 2, fails = 3, score;
…

switch (score) {
    case 9: case 10:  ++a_grades; break;
    case 8:           ++b_grades; break;
    case 7:           ++c_grades; break;
    default:          ++fails;
 }
```

Expressions

BS p. 120
p. 345: operator precedence and associativity
p. 366: Arthmetic Expressions, Relational, Equality and Logical Expressions
p. 365: Autoincrement/decrement, sizeof operator
p. 366: Bit-Manipulation Expressions

Conditonal Expression

x= (y<z) ? y : z;

expr1 ? *expr2* : *expr3*

Comma Expression

expr1 , *expr2*

not in Java

Evaluate *expr1* then *expr2*.
Has value and type of *expr2*.

- x= (++n, n+2)
- **int** i,j;
  for (i=0, j=10; i+j<=20; ++i,++j)
  {cout << i << " " << j << endl;}

```
0 10
1 11
2 12
3 13
4 14
5 15
```

C++ : Some Expressions Platform Dependent

>> RightShift Bits operator:

Java: >> shift bits right, fills with sign bit
>>> shift bits right, fills with zero **NOT in C++ !!**

C++: >> shift bits right, fills with 0/1 , platform dependent !

integer division / :

Java: integer division truncates toward zero: -23/4 yields -5

C++: the result of integer division involving negative numbers is platform dependent. -23/4 yields -5, -6 ?

C++ standard requires : a= = (a / b)∗b + a%b

Simple Types

Java	type	values
	boolean	true, false
	char	16-bit unicode
	byte	signed 8 bit
	short	signed 16-bit
	int	signed 32 bit
	long	signed 64 bit
	float	32 bit floating point
	double	64 bit floating point

C++	type	modifiers
	char	signed, unsigned
	wchar_t	
	int	short, long, signed, unsigned
	float	
	double	long

char ≡ signed char
int ≡ signed int
short ≡ signed short int
long ≡ signed long int

sizeof(*expression*), sizeof(*typename*):
gives the number of bytes used to represent values of the given type.

sizeof(*typename*): **platform dependent !!**

Ansi C++ requires:

1=sizeof(char)≤ sizeof(short)≤ sizeof(int)≤ sizeof(long)
1≤ sizeof(bool)≤ sizeof(long)
sizeof(char)≤ sizeof(wchar_t)≤ sizeof(long)
sizeof(float)≤ sizeof(double)≤ sizeof(long double)
sizeof(N)=sizeof(signed N)= sizeof(unsigned N), N∈ {char,short int,long int}

Simple Types

sizeof(*typename*): **platform dependent !!**

Typical 32 bit platform

type	values	
unsigned char	0 to 255	8 bit
char	-128 to 127	8 bit
(signed) int	-32 768 to 32 767	16 bit
unsigned int	65 535	16 bit
(signed) short int	-32 768 to 32 767	16 bit
unsigned short int	65 535	16 bit
(signed) long int	-2147483648 to 2147483647	32 bit
unsigned long int	0 to 4 294 967 295	32 bit
float	6 digits of precision	32 bit
double	10 digits of precision	64 bit
long double	10 digits of precision	64 bit

The header file <limits> (or <limits.h>) defines constants defining the integral values representable on a given system:

```
#define CHAR_BIT        8               /* number of bits in a char */
#define SCHAR_MIN       (-128)          /* minimum signed   char value */
#define SCHAR_MAX       127             /* maximum signed   char value */
#define UCHAR_MAX       255             /* maximum unsigned char value */
#define SHRT_MIN        (-32767-1)      /* minimum signed   short value */
#define SHRT_MAX        32767           /* maximum signed   short value */
#define USHRT_MAX       65535U          /* maximum unsigned short value */
#define LONG_MIN        (-2147483647L-1)    /* minimum signed   long value
#define LONG_MAX        2147483647L         /* maximum signed   long value
#define ULONG_MAX       4294967295UL        /* maximum unsigned long value
…
```

The range of floating point values are defined in header file <float>

Declarations and initialisation

type *id = expression*

```
void f(int n)
{
  int n;                // value of n undefined
  int i= 5;             // i is initialised to 5
  char c1, c2='B';      // c1 is uninitialised
  double  x= 0.777, y= x+i;
  …
}
```

Initialisation can involve an arbitrary expression, provided that all of the variables and functions used in the expression are defined

Constants/Read-Only Variables

```
const int c1= 5, c2=10;     // constants MUST be initialised
const double Cr= my_f(3.2);  // don't know the value of Cr at compile time
c2= 20;     // illegal !!
```

Typedef-declarations

```
double miles;           // miles a variable
```

```
typedef double miles;    // miles a typename, miles ≡ double
miles distance= 5.2;     //a variable of type double
```

Enumeration Types

An enumeration type declaration:

enum suit { clubs, diamonds, hearts, spades};// range: 0-3

⇒ suit: a new integer type (a *tag name*)

clubs, diamonds, hearts, spades : integer constants

club=0, diamonds=1, hearts= 2, spades= 3

Not in Java

int a;

suit S= diamonds;

S= clubs;
a= hearts; //legal, hearts converted to integer 2
S= 2 // illegal
S= **static_cast**<suit>(2); //explicit cast, ok

Enumerators can be initialised to arbitrary integer constants:

enum Myconsts {A, B=3,C,D,E=10,F,G} // range 0-15

⇒ Myconsts: a new integer type (a *tag name*)

A,B,C,D,E,F,G: integer constants

A=0,B=3,C=4,D=5,E=10,F=11,G=12

Myconsts X= F;
X= C;
X= **static_cast**<Myconsts>(7); // legal, within range
X= **static_cast**<Myconsts>(20); // result undefined, outside range

enum pet {cat, dog} mypet= cat; //Define enumtype and variable

typedef enum {cat, dog} pet; //Define enumtype pet

A Quick view of C-arrays and C-strings

double A [100]; // A an array of 100 elements of type **double**, 0-99
…

double sum= 0.0;
for (**int** i=0; i<100; ++i) {sum+= A[i];}

Java notation:
double[] A= **new** ...

Illegal C++

int B[5]= {10,11,12,13,14}; // array definition with initialisation, 0-4

double R [ ]= {3.2, 4.5, 5.6, 7.8}; // array with 4 **double**'s, 0-3

Two Dimensional Array

int M[ ][3]= {{1,2,3},{4,5,6},{7,8,9},{10,11,12}}; //~ **int** M[4][3]
…

for(**int** i=0; i<4; ++i)
for(**int** j=0; j<3; ++j){cout << M[i][j] << ", ";}

C-string: a null terminated array of char's

char S [ ]= {' a' ,' b' ,' **0**' }; d' ,'
char T [ ]= "abcd";

cout<< S << endl; // print: abcd
cout << T << endl; // print: abcd

C-array's, No Index Check

int A[10];
for (**int** i=0; i<20; ++i){A[i]= 2*i; } // no index-check!!, disaster!