McGill

COMP 208 – Computers in Engineering

COMP 208
Computers in Engineering

Lecture 24

Jun Wang
School of Computer Science
McGill University
Fall 2007

Separate compilation

- When program gets big, it makes sense to divide it into smaller pieces known as modules – divide and conquer.
- A modular program is easier to read/understand/update.
- It also shortens compilation time because only the module that has been changed needs recompile.
- A C programmer usually puts a module in a separate source file, known as a translation unit.

Example

foo.c

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

void foo()
{ printf("foo= %d\n", FOO);}

void bar(int i)
{ printf("bar: %d\n", i); }
```

foo.h

```
#define FOO 100
void foo();
void bar();
```

myprog.c

```
#include "foo.h"

int main()
{
  int x = FOO + 1;
  foo();
  bar(x);
}
```

```
// Separate compilation; -c means comile
// only, do not link; -o specifies output
// file name

gcc –c foo.c  // creates foo.o
gcc –c myprog.c // creates myprog.o
gcc –o myprog myprog.o foo.o  //link
```

Command-line arguments

```
int main(int argc, char* argv[])
{
...
}
```

- argc is the number of command-line arguments, including program name
- argv is the array of arguments as strings, the 1st (argv[0]) being the name of the program

examples

```
#include <stdio.h>

int main(int argc, char* argv[])
{
  int i;
  printf("argc= %d\n", argc);
  printf("agruments are:\n");
  for(i=0; i<argc; i++)
    printf("%s\n", argv[i]);
}
```

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

int main(int argc, char* argv[])
{
  if(argc != 2) {
    printf("Usage: %s <number>\n", argv[0]);
    exit(1);
  }
  printf("number = %d\n", atoi(argv[1]));
}
```

McGill

COMP 208 – Computers in Engineering

Final review

Fortran vs. C: basics

	Fortran	C
Case sensitivity	insensitive	sensitive
Variable declaration	implicit none	variables must be explicitly declared
Comments	one-line comments from ! to end of line	one-line comments starting with // block comments between /* and */
statement termination	statements terminated by end-of-line or keyword	; or } if it's a block

Program structure

FORTRAN program:

```
PROGRAM program-name
  IMPLICIT NONE
  {declarations}
  {statements}
END PROGRAM program-name
  {subprogram definitions}
```

- C program is a collection of functions, one of which has the name main.

Built-in data types

- FORTRAN
  - INTEGER
  - REAL
  - LOGICAL
  - CHARACTER
- C
  - int, long
  - float, double
  - no logical type
  - char
  - type of string is char*

Variable declarations

- FORTRAN

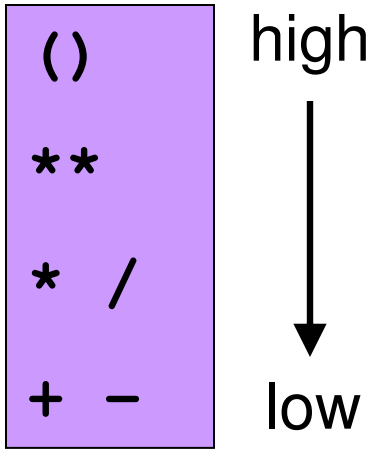
```
INTEGER :: day
INTEGER :: month, year
INTEGER :: hour = 15, minute
REAL :: x, y, z
```

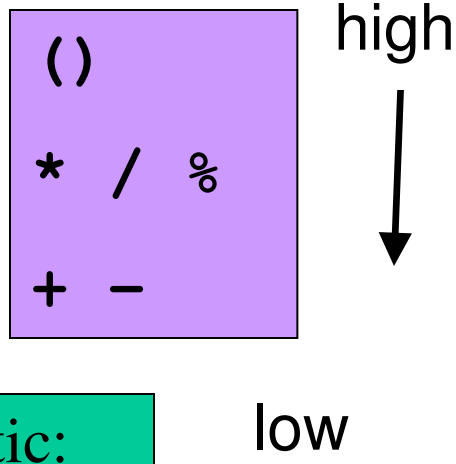
- C

```
int day;
int month, year;
int hour = 15, minute;
double x, y, z;
```

Operators

- Arithmetic operators
  - FORTRAN
    - + - * / **
    - mod is provided by the function mod.
  - C
    - + - * / %
    - no exponentiation





integer arithmetic:
10/4 -> 2
3/5 -> 0
same in both languages

Operators

- Relational operators
  - FORTRAN

    > < >= <= == /=
  - C

    > < >= <= == !=
- Logical operators
  - FORTRAN

    .NOT. .AND. .OR.

    .EQV. .NEQV.
  - C

    ! && ||

Assignment operator

- FORTRAN

  `var = exp`

  – in Fortran, assignment is an operation, and it cannot be used as an expression

- C

  `var = exp;`

  – compound assignment

  a += 3;

  b *= c + 1;

  – In C assignment in addition to being an operation also has values, and it can be used as a sub-expression inside another expression

```
if ((a = b+c) > d)
  f++;
```

this would be illegal in Fortran

```
if (a = 0)
  f++;
```

f++ is never executed

Output

- FORTRAN
  - WRITE (*,*) a, b, c
    - 1st parameter is file number, * means default, screen
    - 2nd parameter: format string, or label of a format statement; * means compiler will determine the format
  - WRITE(*,*) prints an empty line

- C: output provided by library functions
  - printf("%d%f%d\n", a, b, c);
    - 1st parameter is format string; rest are values
  - printf("\n"); prints an empty line

Input

- FORTRAN
  - READ (*,*) a, b, c
    - 1st parameter is file number, * means default, keyboard
    - 2nd parameter: format string, or label of a format statement; * means compiler will determine the format

- C: input provided by library functions
  - scanf("%d%f%d\n", &a, &b, &c);
    - 1st parameter is format string; rest are values

Conditional execution (selection)

- FORTRAN
  - logical IF

```
IF (logical_exp) single-statement
```

```
IF (x > y) min = y
```

  - IF-THEN-ENDIF

```
IF (logical-exp) THEN
  statement block s1
END IF
```

  - IF-THEN-ELSE-ENDIF

```
IF (logical-exp) THEN
  statement block s1
ELSE
  statement block s2
END IF
```

logical exp can be values .TRUE., .FALSE., or relational expression

Conditional execution (selection)

- FORTRAN
  - multi-branch selection

```
IF (logical-exp, e1) THEN
   statement block, s1
ELSE IF (logical-exp, e2) THEN
   statement block, s2
ELSE IF (logical-exp, e3) THEN
   statement block, s3
  . . . . . .
ELSE
   statement block, se
END IF
```

```
IF (x > 0) THEN
  WRITE (*,*) "x is positive"
ELSE IF (x < 0) THEN
  WRITE (*,*) "x is negative"
ELSE
  WRITE (*,*) "x is 0"
END IF
```

Conditional execution (selection)

- C
  - if

```
if(logical-exp)
  single-statement;
```

```
if(logical-exp)
{
  statement block;
}
```

  - if-else

```
if (logical-exp)
{
  statement block s1
}
else
{
  statement block s2
}
```

each block statement can be a single statement

loops

- FORTRAN
  - definite iterator

```
DO var = initial-value, final-value, step-size
  statement block, s
END DO
```

- control variable must be INTEGER
- if step-size is 1, it can be omitted

```
DO i = 1, 10, 2
  WRITE (*,*) i
END DO
```

What's the value of i after the loop?

```
DO i = 7, 0, -3
  WRITE (*,*) i
END DO
```

What's the value of i after the loop?

loops

- FORTRAN
  - indefinite iterator

```
DO
   statement-block, s
END DO
```

  - loop body must contain EXIT statement

```
i = 1
DO
   IF (i > 10) EXIT
   WRITE (*,*) i
   i = i + 2
END DO
```

loops

- FORTRAN
  - DO-WHILE

```
DO WHILE (logical-expression)
   statement-block, s
END DO
```

  - This is equivalent to

```
DO
   IF (.NOT.(logical-expression)) EXIT
   statement-block, s
END DO
```

loops

- C
  - For-loop

```
for (init; condition; increment)
  single-statement or block statement
```

    - initialization part is done only once, before the 1st iteration
    - condition is checked, and if true, loop body is executed, followed by increment part
    - any one of the 3 parts can be empty
  - While-loop

```
while (condition)
  single-statement or block statement
```

loops

- C
  - Do-while-loop

```
do {
  block statement
} while (condition)
```

    - condition check after loop body has been executed
  - break statement
  - continue statement

Arrays

- FORTRAN

```
type :: name(bound)
```

```
type :: name(m, n)
```

```
INTEGER :: arr(10)
INTEGER :: table(3, 5)

arr[2] = 23;
table(1, 3) = 1;
```

  - index starts at 1

- C

```
type name[bound];
```

```
type name[m][n];
```

```
int arr[10];
int table[3][5];

arr[2] = 23;
table[1][3] = 1;
```

  - index starts at 1
  - can have initializer

Array input in FORTRAN (lecture 8, 9)

- 3 ways to input data to an array:

```
REAL :: A(1000)
...
DO I = 1, SIZE
    READ (*,*) A(I)
END DO
```

- accepts only 1 value per line
- accepts values for part of the array, or entire array

```
REAL :: A(1000)
...
READ (*,*) (A(I), I=1,SIZE)
```

- accepts multiple values per line
- accepts values for part of the array, or entire array

```
REAL :: A(1000)
...
READ (*,*) A
```

- accepts multiple values per line
- accepts values for entire array only

Array output in FORTRAN

- output an array:

```
REAL :: A(1000)
...
DO I = 1, SIZE
    WRITE (*,*) A(I)
END DO
```

write one element per line

```
REAL :: A(1000)
...
WRITE (*,*) (A(I), I=1,SIZE)
```

write elements 1 to size on one line

```
REAL :: A(1000)
...
WRITE (*,*) A
```

write all elements on one line