Software Tools

C, Unix (Linux), and tools

Precedence and Order of Evaluation

- Precedence has to do with associativity
  - In `a+b*c` is executed as `a+(b*c)`
- Order of evaluation has to do with what is executed first
  - In `i<MAXARR && V[i]!=0` the first clause is evaluated first
- C has a long table of associativities
- C specifies order of evaluation for some operators only:
  - `&& || ?:` and ‘,’

Associativity

- The golden rule
  - When in doubt parenthesize
- A few things to remember
  - Assignment operators have very low precedence
  - Unary operators have high precedence
  - Arithmetic operators have generally higher precedence than logical or relational ones

Order of Evaluation

- In most cases is not specified
  - Different compilers and different architectures behave differently
- Statement `f(n++, A[n])` or `n++ * A[n]` can produce different results at different times
- It is bad practice to depend on the order of evaluation (other than for AND and OR)

Compound Statements

- Statements inside { and } form a block aka compound statement
- Single statements end with a semicolon ;
  - A semicolon turns an expression into a statement
- There is no ; needed after a right brace.
- Semicolons *end* statements, do not separate them

If-else

- Typical if-else statement
- The thing to remember that since the else is optional there is an ambiguity:

```
c=0;
if a<o
  if b>0
    c=a+b;
  else
    c=a-b;
```

Switch-case

- A multiway decision to test if an expression matches one of the constant integer valued labels (cases)
- The cases serve as labels to “blocks”
  - The blocks do not need braces
  - They are *fall through*
  - To avoid *fall through* we use breaks

- This counts small digits and all digits

```
while ((c=getchar()) != EOF) {
  switch (c) {
  case '0': case '1': case '2': case '3': case '4':
    Nsmall++;
  case '5': case '6': case '7': case '8': case '9':
    Ndigit++;
    break;
  default:
    printf("Not a digit: %c\n", c);
  }
}
```

Loops

- For loop
- While loop
- Do while loop

```
while (expr1)
  Stmnt;
```

```
for (expr1; expr2; expr3)
  Stmnt;
```

```
do
  Stmnt;
while (expr);
```

Break and continue

- A very clean way to get out the middle of a loop
- With break we get out of the loop immediately
- With continue we go back to the beginning of the iteration
- Break is used quite often
- Continue, not so often

Trim function

```
int trim(char s[])
{
  int n;

  for (n=strlen(s)-1; n>=0; n--)
    if (s[n]!=' ' && s[n]!='\t' && s[n]!='\n')
      break;
  s[n+1] = '\0';
  return n;
}
```

Goto

- The `goto` statement was popular until the 70's
- Code using `goto`s is easy to write, hard to write correctly and very hard to debug.
- The fashion to abolish it was called *structured programming*
- In reality it is very rarely needed if at all.

```
if (a[i]<0) goto errorlbl;
return sqrt(a[i]);
errorlbl: printf("...");
return 0;
```