Software Tools

C, Unix (Linux), and tools

Arrays

- An easy intro to arrays
- Arrays in C are really pointers
  - But do not worry about it for now.
- We see how we define fixed size arrays
- We see how we assign values to them

The wordcnt Prog.

```
#include <stdio.h>

int main()
{
  int c, i;
  int ndigit[10];

  for (i=0; i<10; i++) * Set all elements to zero *
    ndigit[i]=0;

  while ( (c=getchar()) != EOF )
    if ('0'<=c && c<='9')
      ndigit[c-'0']++;
  printf("# of digits: ");
  for (i=0; i<10; i++)
    printf(" %d", ndigit[i]);
  printf("\n");
}
```

Functions

- See how we declare functions
- How we define functions
- How we return values
- How we specify the parameters (the variables that appear in the function header)
- How we pass arguments (the values that we give to a function when we invoke it)

The power function.

```
#include <stdio.h>

int power(int m, int n);   /* function declaration */
                           /* aka function prototype */
int main()
{
  /* .... */
  return 0;
}

int power(int base, int n)    /* function definition */
{
  int i, p;

  for (i=1, p=1; i<=n; i++)
    p *= base;
  return p;
}
```

Arguments: Call by value

- In C arguments are copied to the function
- So if we provide a variable then the function gets a copy of this variable
  - This means that if the function modifies this parameter, it modifies only the copy, not the variable itself
- A seeming exception is arrays
  - Arrays in C are pointers (more on this later)
- In the modified power function in the next slide the caller does not see the changes

The new power function.

```
#include <stdio.h>

int power(int m, int n);   /* function declaration */
                           /* aka function prototype */
int main()
{
  /* .... */
  return 0;
}

int power(int base, int n)    /* function definition */
{
  int p;

  for (p=1; n>0; n--)
    p *= base;
  return p;
}
```

Arrays of Characters

- Aka strings (if `NULL` terminated)
- The char type is one byte long
- Such arrays are terminated by a null character
- The null character is `'\0'` which is the same as `0`.
- So a string with 5 characters is at least 6 elements long.
- C does not know/care/check array sizes
  - That’s the job of the programmer

Defining Arrays

- For now we care about arrays of constant size.
  - e.g. `char line[1000];`
- If the array is defined inside a function the array exists while the function is alive.
  - The data in the array can be modified by that function and any function that receives the array as argument.
- If it is defined outside any function the array exists while the program is alive.
- In both cases at least 1000 elements are available.
- If we try to access/modify the 1001 element then bad things will happen **only** if a boss/grader is nearby.

The getline function.

```
#include <stdio.h>

int KRgetline(char s[], int lim)
{
  int c, i;

  for (i=0; i<lim-1 && (c=getchar())!=EOF && c!='\n'; i++)
    s[i] = c;
  if (c=='\n') {
    s[i] = '\n';
    i++;
  }
  s[i] = '\0';
  return i;
}
```

The copy function.

```
#include <stdio.h>

void copy(char to[], char from[])
{
  int i;

  for (i=0; (to[i]=from[i]) != '\0'; i++);
}
```

External variables

- All these variables we defined were available in the function we defined them in (unless we pass them as arguments)
- We can also define them outside any function and make them available to all functions as global variables.
- Most programs need some global variables. But
  - Global variables are a source of tears (hard to debug)
  - Use them only if absolutely necessary

The main (function/program).

```
#include <stdio.h>
#define MAXLINE 1000            /* max line size */

int  KRgetline(char line[], int maxline);
void copy(char to[], char from[]);

char line[MAXLINE], longest[MAXLINE];

int main()
{
  int len, max;
  /*  extern char line[], longest[];*/

  max = 0;
  while (...
}
```

Problems to play with

- Write a function that reads from the standard input a line at a time and prints out the characters of the line in reverse order.
- Write a program that checks if parentheses are balanced. The program uses a variable `cnt` that it is incremented when a left parenthesis is encountered and decremented when a right one is encountered. The `cnt` should be always positive or zero and at the `EOF` should be zero.