Computer programming in INF236

Today:

- Introduction to C
- Some important differences from Java

The C programming language

Developed 1969 – 1973 by Dennis Ritchie
"High level assembly code"

Not object oriented! (Look at C++)

Similar types of constructions as Java
- Declaration of variables (int, float, double, boolean etc)
- Code blocks delimited by {…}
- While and for loops just like Java
- Use of (static) procedures and functions

Some differences:
- No objects or classes
- Compiles to assembly code, no portable byte code
- Gives more low level control, often more efficient code
- Use of pointers
- Allocation of memory
- Input – Output (simpler than in Java)

Programming in C

The simplest and best known among C programs:

```
#include <stdio.h>

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

  printf("Hello World!\n");

}
```

Compile with:

```
gcc hello.c -o hello
```

Run with:

```
./hello
```

Will result in:

```
Hello World!
```

Pointers

```
int i=5;
int *a;        // The * says that a is a pointer to an int
a = &i;        // Now a points at i

printf("The variable i has value %d.\n", i); // Should print 5

// Should also print 5:
printf("The pointer a points to an int with value %d.\n", *a);
```

Observations:

`&i` returns the memory address of variable i

`*a` returns the value of the variable (memory cell) to which a points

Pointers are useful:

to build arrays using dynamic memory allocation

when passing arguments to functions

Dynamic memory allocation

One-dimensional arrays:

```
// Static memory allocation:
int aStatic[100];
aStatic[5] = 73;

// Dynamic memory allocation:
int n=100;                          // Array size
int *a;                             // To point at the FIRST array element
a = (int*) malloc(n*sizeof(int));   // malloc allocates contiguous
                                       memory!

// Now a can be considered as an array:
int i;                              // Counters cannot be declared within for
                                       statements!

for(i=0;i<n;i++)
  a[i] = rand();                    // Fill a with random integers

for(i=0;i<n;i++)
  printf("a[%d]=%d\n",i,a[i]);

free(a);                            // Give back what you borrowed!
```

Dynamic memory allocation

Observations:

`malloc(m)`

- allocates `m` bytes of contiguous memory
- returns the memory address (`void*`) of the first byte
- must cast returned memory address to wanted type

`sizeof(datatype)` = number of bytes occupied by `datatype`-variables

When the memory is no longer needed it must be released (using `free`) to avoid memory leaks.

The compiler (might) accept:

```
int *a;
a[5] = 73;
```

but a runtime error (segmentation fault) could occur.

You can write either `int *a` or `int* a`
You can write either `a[i]` (recommended) or `*(a+i)`

Note that there is no range checking at runtime.

Dynamic memory allocation

Two-dimensional arrays:

```
// Static memory allocation:
int aStatic[100][200];
aStatic[5][8] = 73;

// Dynamic memory allocation:
int m=100, n=200;          // Array size (rows, columns)
int **a;                   // Pointer to an int-pointer
a = (int**) malloc(m*sizeof(int*)); // space for row pointers

// Allocate memory for each row
int i,j;                   // Row and column counters
for(i=0;i<m;i++)
  a[i] = (int*) malloc(n*sizeof(int)); // Rows have length n
                                       // No default value!

for(i=0;i<m;i++)
  for(j=0;j<n;j++)
    a[i][j] = rand();      // Fill a with random integers
```

Dynamic memory allocation

Two-dimensional arrays:

```
for(i=0;i<m;i++)
  for(j=0;j<n;j++)
    a[i][j] = rand();  // Fill a with random integers

// … Do something with this array

for(i=0;i<m;i++)
  free(a[i]); // Free memory allocated to row i
free(a);  // Free memory allocated to the row pointers
```

Observations:

- k-dimensional `int`-arrays can be declared as `int *...*a` (k asterisks)
- requires nested loops (k-1 levels) of calls to `malloc` and `free`

Dynamic memory allocation

Two-dimensional arrays in contiguous memory:

```
// Dynamic memory allocation:
int m=100, n=200;          // Array size (rows, columns)
int **a;                   // Our array (almost) as before
int *p;                    // Auxiliary pointer
p = (int*) malloc(m*n*sizeof(int));   // m*n integers
                                      // in contiguous memory

// Allocate memory for row pointers:
a = (int**) malloc(m*sizeof(int*));

// Assign value to each row pointer:
int i,j;
for(i=0;i<m;i++)
  a[i] = p+i*n;            // Move i rows beyond the start of a

// Go on as before...
for(i=0;i<m;i++)
  for(j=0;j<n;j++)
    a[i][j] = rand();
```

Dynamic memory allocation

Two-dimensional arrays in contiguous memory:

```
// Go on as before...
for(i=0;i<m;i++)
  for(j=0;j<n;j++)
    a[i][j] = rand();

// Releasing memory:
free(a);
free(p);
```

Passing parameters to functions

```
void myFunction(int u, double v);       // Function prototype

int main(int argc, char *argv[]) {
  int a=0;
  double b=0.0;
  myFunction(a,b);
  // What are the values of a and b?

}

void myFunction(int u, double v) {
  u = 1;
  v = 3.14;
}
```

Passing parameters to functions

```
void myFunction(int u, double v);       // Function prototype

int main(int argc, char *argv[]) {
  int a=0;
  double b=0.0;
  myFunction(a,b);  // Call by value (just like in Java)
  // What are the values of a and b?
  // Answer: Still a=0 and b=0.0
}

void myFunction(int u, double v) {

  u = 1;

  v = 3.14;

}
```

Passing parameters to functions

```
void myFunction(int *u, double *v); // New function prototype


int main(int argc, char *argv[]) {
  int a=0;
  double b=0.0;
  myFunction(&a,&b);
  // What are the values of a and b?

}

void myFunction(int *u, double *v) {
  *u = 1;
  *v = 3.14;
}
```

Passing parameters to functions

```
void myFunction(int *u, double *v); // New function prototype


int main(int argc, char *argv[]) {
  int a=0;
  double b=0.0;
  myFunction(&a,&b);  // Call by reference (not in Java)
  // What are the values of a and b?
  // Answer: Changed to a=1 and b=3.14
}

void myFunction(int *u, double *v) {
  *u = 1;
  *v = 3.14;
}
```

Passing parameters to functions

```
void swap(double *u, double *v) {
  double tmp=*u;
  *u=*v;
  *v=tmp;
}
```

Use:

```
  int n=100;
  double *a;
  a = (double*)malloc(n*sizeof(double));
  // … fill a with real numbers
  // … let i and j be integers in 0..99
  swap(a+i, a+j);           // Swap a[i] and a[j]
  swap(&(a[i]), &(a[j]));  // Would give the same result
```

Passing arrays to functions

```
void initialize(double *a, int size) {
  int i;
  for(i=0;i<size;i++)
    a[i] = 0.0;
}
```

Use:

```
  int n=100;
  double *a;
  a = (double*) malloc(n*sizeof(double));
  intialize(a,n); // note that a is a pointer to a double
```

Keyboard and file input

Let the user assign values to an `int` and a `double`:

```
int n;
double f;
printf("Enter an integer and a real number: ");
scanf("%d %lf", &n, &f);
```

Read an int and a double from the file myData.txt

```
int n;
double f;
// Open the file in input (r) mode
FILE *filePtr = fopen("myData.txt", "r");
if (filePtr)
  fscanf(filePtr, "%d %lf", &n, &f);
else
  printf("Could not open myData.txt for reading.\n");
```

File output

Save results in the textfile `myData.txt`

```
int n=100;
double f=3.14;
// Open the file in output (w) mode
FILE *filePtr = fopen("myData.txt", "w");
if (filePtr)
  fprintf(filePtr, "%d %lf", n, f);
else
  printf("Could not open myData.txt for writing.");
```

Getting Help

Number of online C tutorials
https://en.wikibooks.org/wiki/C_Programming

Buy a (physical) book

Look up manual pages:

```
>man printf

PRINTF(1)    User Commands         PRINTF(1)

NAME
       printf - format and print data

SYNOPSIS
       printf FORMAT [ARGUMENT]...
       printf OPTION

DESCRIPTION
       Print ARGUMENT(s) according to FORMAT.
```