Basic Principles:
Memory Organization

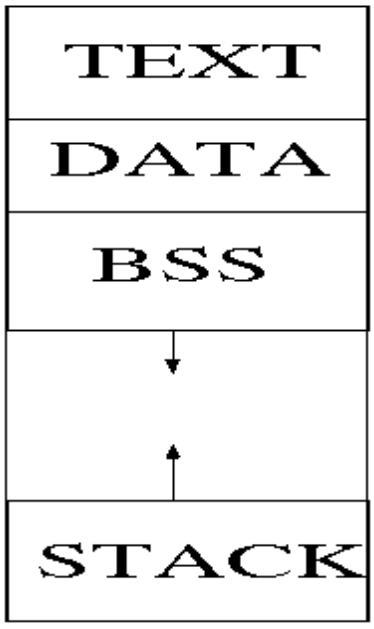
Main Memory is : an ARRAY OF BYTES
Addresses: 0 to 0 xFFFFFFFF
(machine dependent)

A Pointer is : AN ADDRESS IN MEMORY

Data is : contents of some area in memory.

Code is : contents of some area in memory.

(Virtual) MEMORY MAP of a process



Basic Principles: Stack

(Function call) STACK: a region of memory.

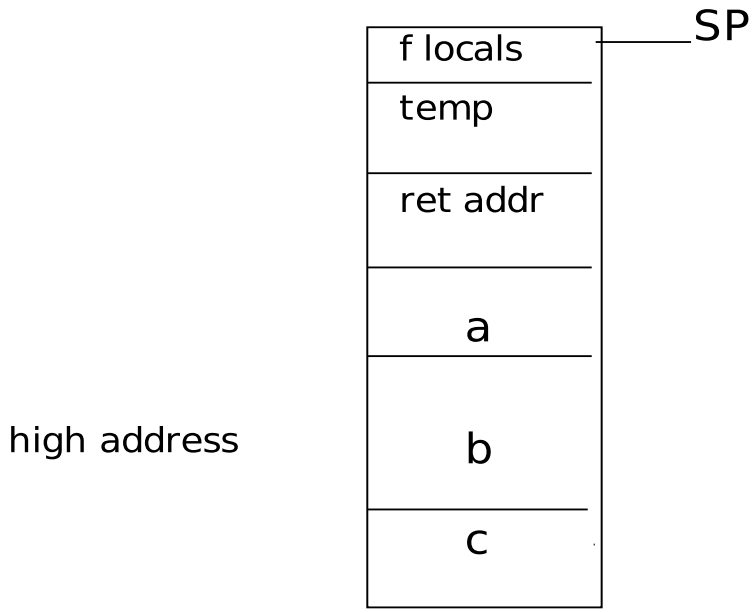
Storage for:

- Local variables
- Function arguments
- Return address

Calling convention: language dependent.

For C: push argument VALUE from right to left.

Example: f(a, b, c);



Basic Principles: CPU Registers

CPU contains some registers:

- Program counter ("pointer")
- General purpose registers
  (used as data or pointers)
- Internal status/control registers

Structure of registers: very machine dependent (more in architecture course).

Structure of memory: (almost) machine independent (as seen from most HLL)

C programming language: allows easy access to "raw" memory.

Basic Principles: Programs

Computer programs (compiled code):
1 Start as high-level language like C.
2 Run compiler to generate binary OBJECT CODE file.
3 Linkage editor ("linker") combines OBJECT and LIBRARY files to create binary EXECUTABLE FILE.
4 Excutable loaded/mapped into main memory by PROGRAM LOADER and can then run.
5 Additions: dynamic linking/loading.

Operating system provides SYSTEM SERVICES to a running program through SYSTEM CALLS.

Basic Principles:
Operating System

NOT same as window manager and/or command interpreter.

Command interpreter is a USER program, can be either:
1 Command-line interpreter.
2 Windows point-and click interpreter.

Operating system provides basic services:
1 Process scheduling
2 Memory management
3 Communications
4 File system
5 Other device access
In UNIX/LINUX most things visible as "files".

C Programming Language

1 Basics.
2 Main differences from Java.
3 Data and storage types, pointers and structures.
4 Functions.
5 Input-output in C (or lack of).

C Language - Basics

Simple example program blah.c:

```
#include <stdio.h>

int i = 1;

main() {

 printf("%d There is no magic\n", i);

}
```

Includes:

- Preprocessor commands
- (Global) Declarations
- Functions

C Language - Basics

Compiling:

gcc blah.c

Creates (eventually) an executable file. (called "a.out" by default).

To learn on your own: command flags (e.g. -o).

Steps:

1 Preprocessor+2-pass compiler

2 Linker (link with C stdlib+init)

For better control over multiple program files (re-learn on your own): make files

C vs. Java

1 Compiled, not interpreted.
2 Useful pre-processor.
3 No "magic" objects.
4 No garbage collection (explicit malloc / free).
5 WEAK type system.
6 Can access (almost) anything using POINTERS.
7 Very simple semantics (direct translation, very efficient).
8 No IO as part of language (!)

C Data and Storage Types

Basic data types: (define before use)

```
int x;
char y;
unsigned char c;
float  BloodyLongVarName ;
double Whatever ;
char * p;
```

Structure definitions and typedefs:

```
    typedef  struct element {
struct element *next;
        int ID;
        char name [NAME_LENGTH];
    } element;

    element my_element, elements[4];

    my_element.ID = 666;
    elements[0].next = &my_element;
```

C Data and Storage Types: Storage

C storage types:

- Global variables:
  Define outside functions.
  Constant memory address.
  Names used across files.
- Local variables:
  Define in functions (at beginning, not middle!).
  Allocated on stack.
- Static variables (NOT like Java)
- Heap (dynamic) storage:
  Allocated by library functions and system calls.

C Data and Storage Types: Pointers

Pointers: contain a MEMORY address.

Definitions:

```
char *p;      /* Pointer to char */
char (*f)(); /* Pointer to function returning char */
int  *f();    /* Function returning pointer to int */
```

On 80X86/LINUX: 32 bit number

Access through a pointer:

```
*p = 3;
Next->ID = 8;
(*f)();
```

Operations on pointers:

```
if(p == q) { exit(0)};
p = p + 1; /* increment by size of... */
```

"Address of" operator:

```
f = &main;
p = &c;
```

C Data and Storage Types: Pointers and Casting

Consider:

```
int i=2;
char c = 5;
float num;
```

"Automatic" conversion:

```
i = c;
```

Forcing conversion casting:

```
i =((float)i)/5 * c;
```

Especially used for pointers:

```
p = (element *) malloc (sizeof (element));
```

Can be used to (deliberately) "cheat":

```
i =  num;
i = *((int *)(&  num));
```

Or even:

```
i = *((int *)(&main));
```

C Data and Storage Types: "Strings"

Strings are NOT a true C data type.

Implemented as: array of char.

```
char my_ str[] = "There is no magic";
```

Convention: NULL TERMINATED string.
(NULL is 0).

Convention used in most standard library functions, such as open, strcmp, etc.

IMPORTANT: "char"s are simply short, 1 byte integers.

They can REPRESENT characters if we so wish, using, e.g. ASCII (the default), or ANY OTHER representation.

Functions in C

All code in C is in some function.
(But one CAN "cheat"...)

main() is the function called by INIT after program is loaded.

A function receives arguments BY VALUE, and (possibly) returns one value.

Function PROTOTYPE:

```
void main(int ac, char *av[]);
```

Types LOOSELY checked.

Functions definition is FLAT.

Functions in C

Arguments to function:

```
foo(a, b, c);
```

Pushes COPY OF VALUES onto stack starting with rightmost.

Called function can access LESS:

```
foo(int a, int b) {
        a=5;    /* Changes LOCAL copy */
        return(a+b);
}
```

Works perfectly OK if it passes the compiler (e.g. in different files).

Variable definitions at BEGINNING of function!

```
foo() {
    int x, y=4;  /* local variables (stack) */
  static int z=3;  /* single storage inst */
  < function code>
}
```

(Lack of) I/O in C

C language has NO defs. for I/O

Use: system calls + stdlib funcs.
(stdlib functions use system calls)

```
int fd, size, count, mode;
    char buf[BUF_SIZE];
   fd = open("filename", flags, mode);
  size = read( fd, buf_addr, count);
  close( fd);
```

Default file descriptors (UNIX):

0: standard input
1: standard output
2: standard error

Also available - stream functions:

```
FILE *f;
f =  fopen("filename ", "rw");
printf("Debug: just before crash?");
fflush(stdout);
```

By default, BUFFERED IO.