
CS 35L- Software Construction Laboratory

Winter 2019

TA: Guangyu Zhou

Course Information

- Assignment 10 Presentation

- Please **upload your slides** to CCLE before your presentation
 - Speakers next class

2/6	Varun Raju	Kathir Ilango	Hashing
	Dong Han		Open MPI and Conjugate Grandient Method

- ~12 minutes talk in class (~15 min for teams)
 - Participation in Q&A
- Assignment 4 due on Feb 9th

C Programming and Debugging

Part II

Outline

- GDB – Debugging (Review)
 - **Issues on C Programming**
 - **Hints for Assignment 4**
-
-

Running

```
# gdb <program> [core dump]
    Start GDB (with optional core dump).

# gdb --args <program> <args...>
    Start GDB and pass arguments

# gdb --pid <pid>
    Start GDB and attach to process.

set args <args...>
    Set arguments to pass to program to
    be debugged.

run
    Run the program to be debugged.

kill
    Kill the running program.
```

Breakpoints

```
break <where>
    Set a new breakpoint.

delete <breakpoint#>
    Remove a breakpoint.

clear
    Delete all breakpoints.

enable <breakpoint#>
    Enable a disabled breakpoint.

disable <breakpoint#>
    Disable a breakpoint.
```

Watchpoints

```
watch <where>
    Set a new watchpoint.

delete/enable/disable <watchpoint#>
    Like breakpoints.
```

<where>

function_name Break/watch the named function.
line_number Break/watch the line number in the current source file.
file:line_number Break/watch the line number in the named source file.

Conditions

break/watch <where> if <condition> Break/watch at the given location if the condition is met.
Conditions may be almost any C expression that evaluate to true or false.
condition <breakpoint#> <condition> Set/change the condition of an existing break- or watchpoint.

Examining the stack

backtrace
where Show call stack.
backtrace full
where full Show call stack, also print the local variables in each frame.
frame <frame#> Select the stack frame to operate on.

Stepping

step Go to next instruction (source line), diving into function.

next

Go to next instruction (source line) but don't dive into functions.

finish

Continue until the current function returns.

continue

Continue normal execution.

Variables and memory

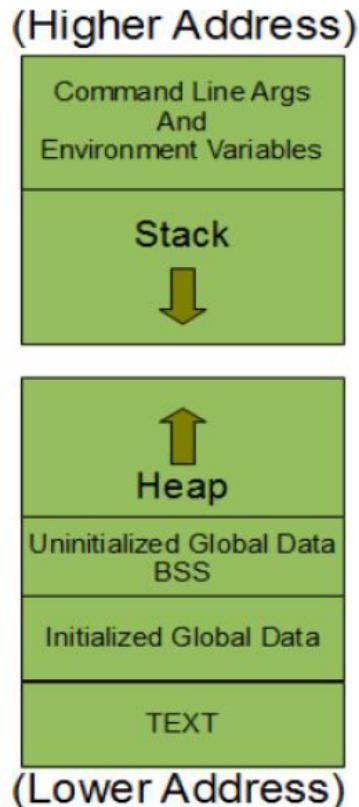
print/format <what> Print content of variable/memory location/register.
display/format <what> Like „print“, but print the information after each stepping instruction.
undisplay <display#> Remove the „display“ with the given number.
enable display <display#>
disable display <display#> En- or disable the „display“ with the given number.
x/nfu <address> Print memory.
n: How many units to print (default 1).
f: Format character (like „print“).
u: Unit.
Unit is one of:

b: Byte,
h: Half-word (two bytes)
w: Word (four bytes)
g: Giant word (eight bytes).

Displaying Source Code

- list *filename: line_number*
 - *Displays source code centered around the line with the specified line number*
- list *line_number*
 - Display lines in the current source file if not specified the file name
- list *from,[to]* (line number/function names)
 - Display the specified range of the source code
- list *function_name*
 - source code centered around the line in which the specified function begins
- list, l
 - Display more lines of source code following those presented by the last list command

Review: Program Process Layout



- TEXT segment
 - Contains machine instructions to be executed
- Global variables
 - Initialized\uninitialized
- Heap segment
 - Dynamic memory allocation
 - Malloc, free
- Stack segment
 - Push frame: function invoked
 - Pop frame: function returned
 - Stores: local variables; return address, registers
- Command line arguments and environmental variables

C Language

- C: a subset of C++, very similar
- Differences
 - No object oriented features: class, overloading, virtual function etc.
 - Memory Allocation
 - For C language: **main function doesn't return 0 automatically**

Type

- Built-in types
 - Integers, Floating-point, character strings
 - No bool (before C99)
- Compiling ‘C’ only
 - `gcc -std=c99 hello.c`

Struct

- No classes in C
- Function: package related data (variables of different types) together
- Single name is convenient (rename: *typedef*)

```
struct Student {  
    char name[64];  
    char UID[10];  
    int age;  
    int year;  
};  
struct Student s;
```

```
typedef struct {  
    char name[64];  
    char UID[10];  
    int age;  
    int year;  
} Student;  
Student s;
```

C Struct vs. C++ Class

- C structs cannot have member functions
- There's no such thing as access specifiers in C
- C structs don't have constructors defined for them
- C++ classes can have member functions
- C++ class members have access specifiers and are private by default
- C++ classes must have at least a default constructor

Pointers Review

- Variables that store **memory addresses**
- Declaration
 - `<variable_type> *<name>;`
 - `int *ptr; //declare ptr as a pointer to int`
 - `int var = 77; // define an int variable`
 - `ptr = &var; // let ptr point to the variable var`

Dereferencing Pointers

- Accessing the value that the pointer points to
- Example:
 - `double x, *ptr;`
 - `ptr = &x; // let ptr point to x`
 - `*ptr = 7.8; // assign the value 7.8 to x`

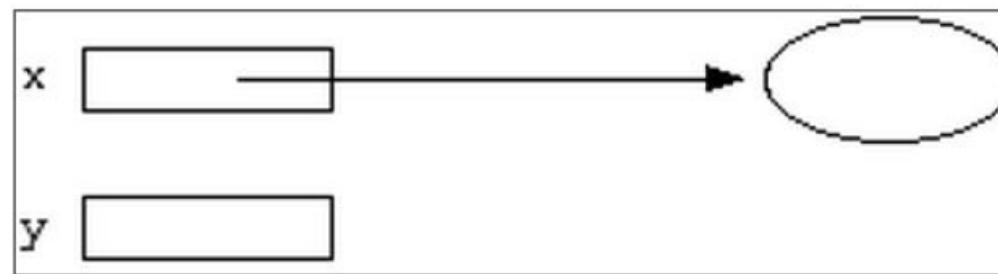
Pointer Example

```
int *x;
```

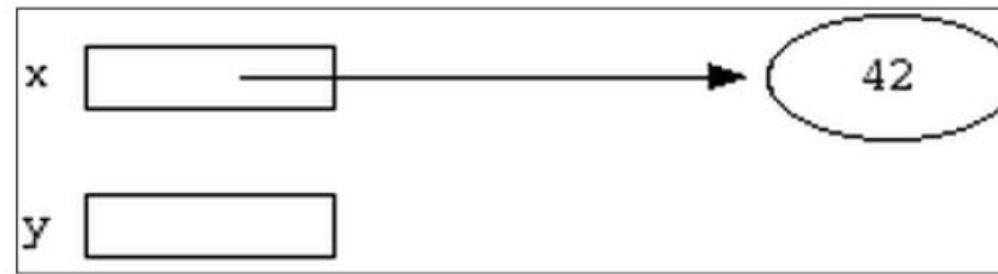


```
int *y;
```

```
int var; x = &var;
```

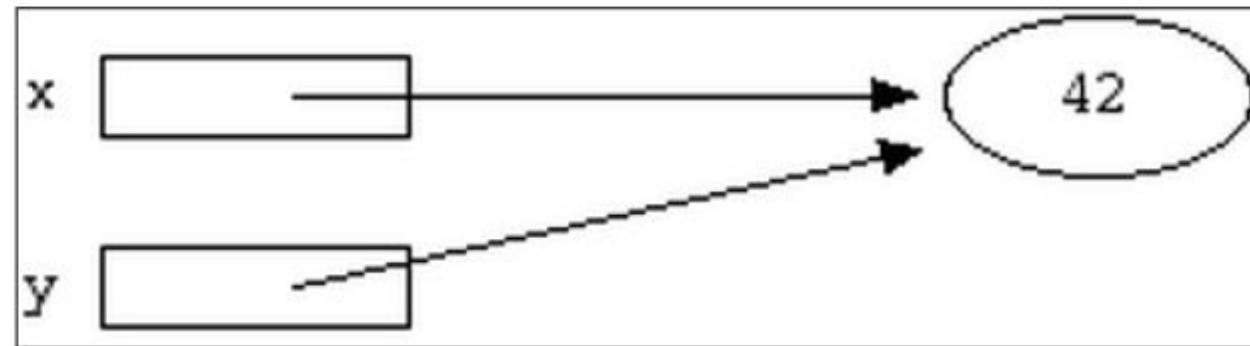


```
*x = 42;
```

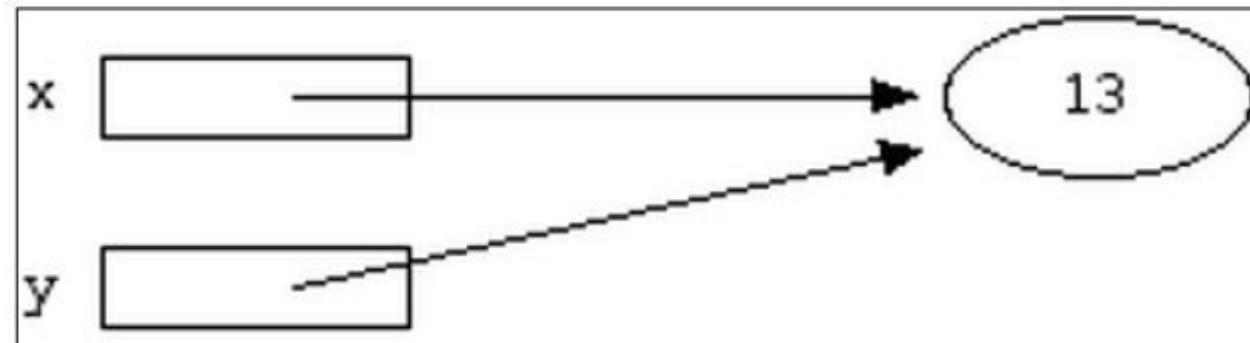


Pointer Example

$y = x;$

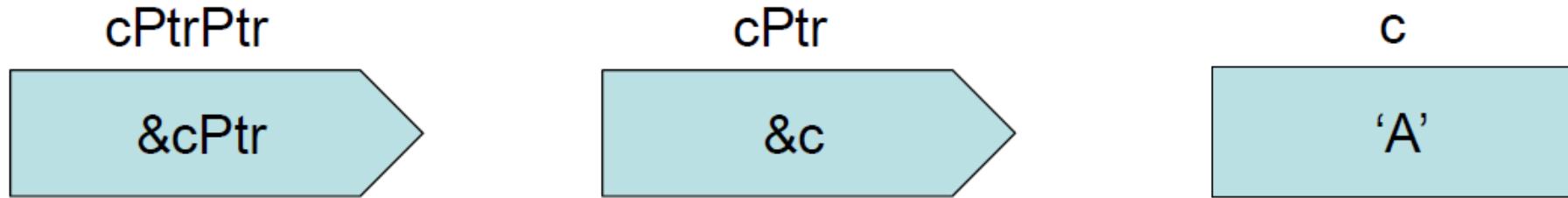


$*x = 13;$ or
 $*y = 13;$



Pointers to Pointers

```
char c = 'A'      char *cPtr = &c      char **cPtrPtr = &cPtr
```



Pointers to Functions

- Also known as: **function pointers** or **functors**
- Task of homework 4: write a sorting function
 - Has to work for ascending and descending sorting + other
- How?
 - Write multiple functions
 - Provide a flag as an argument to the function
 - Polymorphism and virtual functions
 - Use function pointers!!

qsort Example

```
#include <stdio.h>
#include <stdlib.h>
int compare (const void * a, const void * b)
{
    return ( *(int*)a - *(int*)b );
}
int main ()
{
    int values[] = { 40, 10, 100, 90, 20, 25 };
    qsort (values, 6, sizeof(int), compare);
    int n;
    for (n = 0; n < 6; n++)
        printf ("%d ",values[n]);
    return 0;
}
```

Pointers to Functions

- User can pass in a function to the sort function
- Declaration
 - `double (*func_ptr) (double, double);`
 - `func_ptr = &pow; // func_ptr points to pow()`
- Usage
 - `// Call the function referenced by func_ptr`
`double result = (*func_ptr)(1.5, 2.0);`
 - `// The same function call`
`result = func_ptr(1.5, 2.0);`

Dynamic Memory Allocation

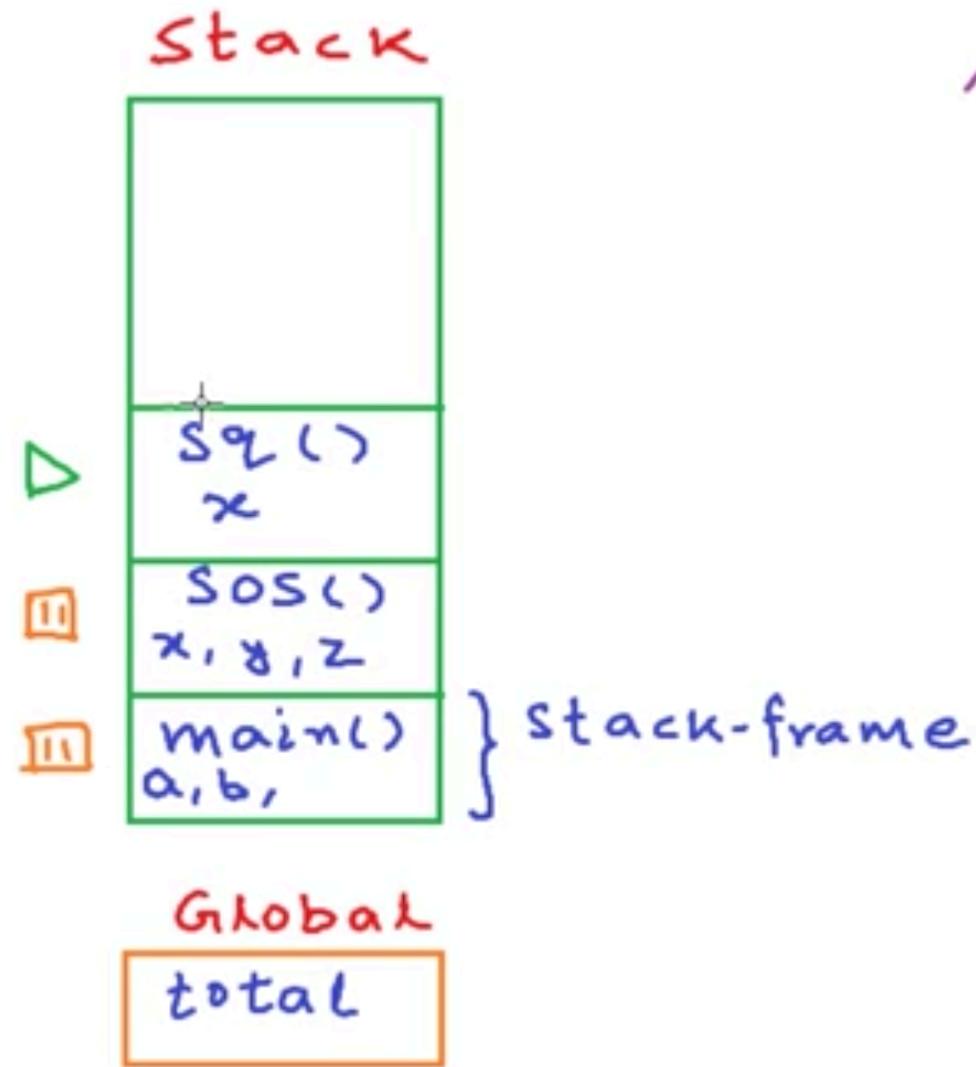
- `malloc(size_t size)`: allocates a block of memory whose size is at least size
- `free(void *ptr)`: frees the block of memory pointed to by ptr
- `realloc(void *ptr, size_t newSize)` : Resizes allocated memory
- Allocate memory **on heap**

```
Rectangle_t *ptr = (Rectangle_t*) malloc(sizeof(Rectangle_t));
if(ptr == NULL)
{
    printf("Something went wrong in malloc");
    exit(-1);
}
else
{
    //Perform tasks with the memory
    //
    //
    free(ptr);
    ptr = NULL;
}
```

```
ptr = (Rectangle_t*) realloc(ptr, 3*sizeof(Rectangle_t))
```

Demo: Stack vs Heap

```
#include<stdio.h>
int total;
int Square(int x)
{
    return x*x; //  $x^2$ 
}
int SquareOfSum(int x,int y)
{
    int z = Square(x+y);
    return z; //  $(x+y)^2$ 
}
int main()
{
    int a = 4, b = 8;
    total = SquareOfSum(a,b);
    printf("output = %d",total);
}
```



Demo: Stack vs Heap

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a; // goes on stack
```

+

Stack



Global

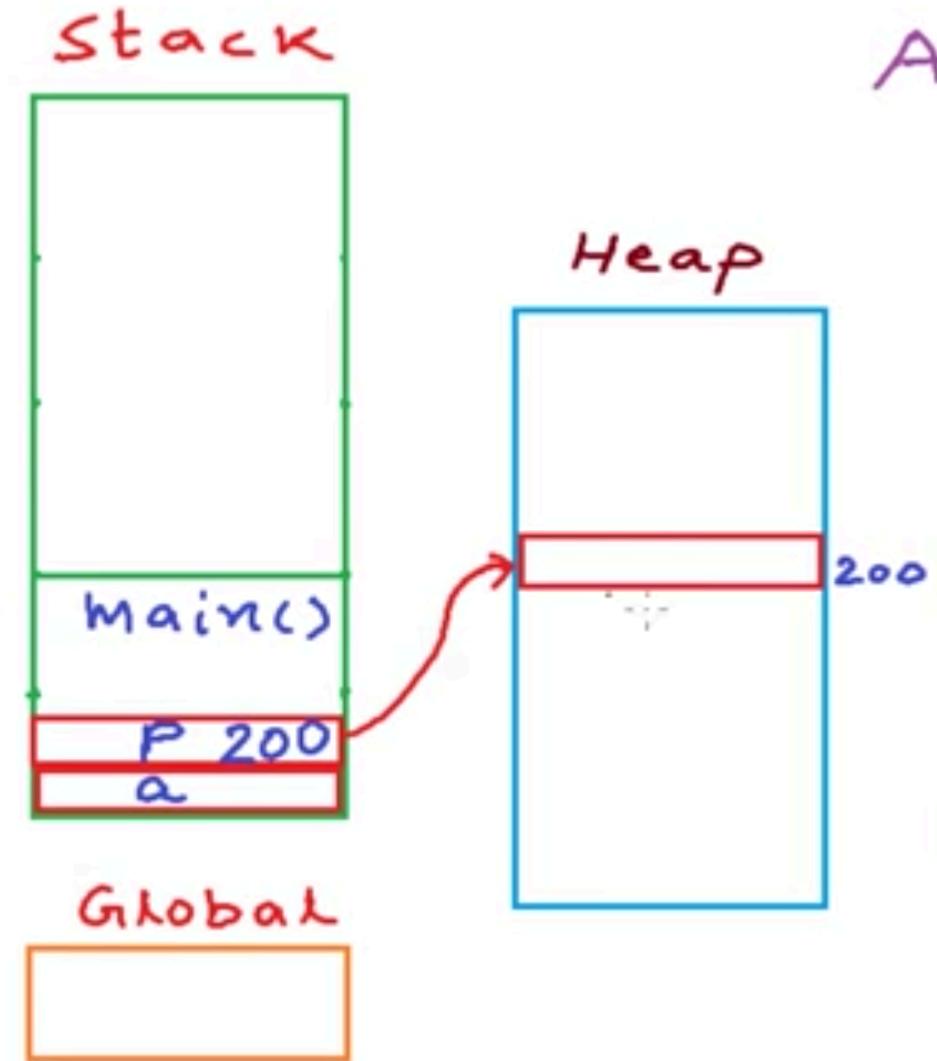


Heap



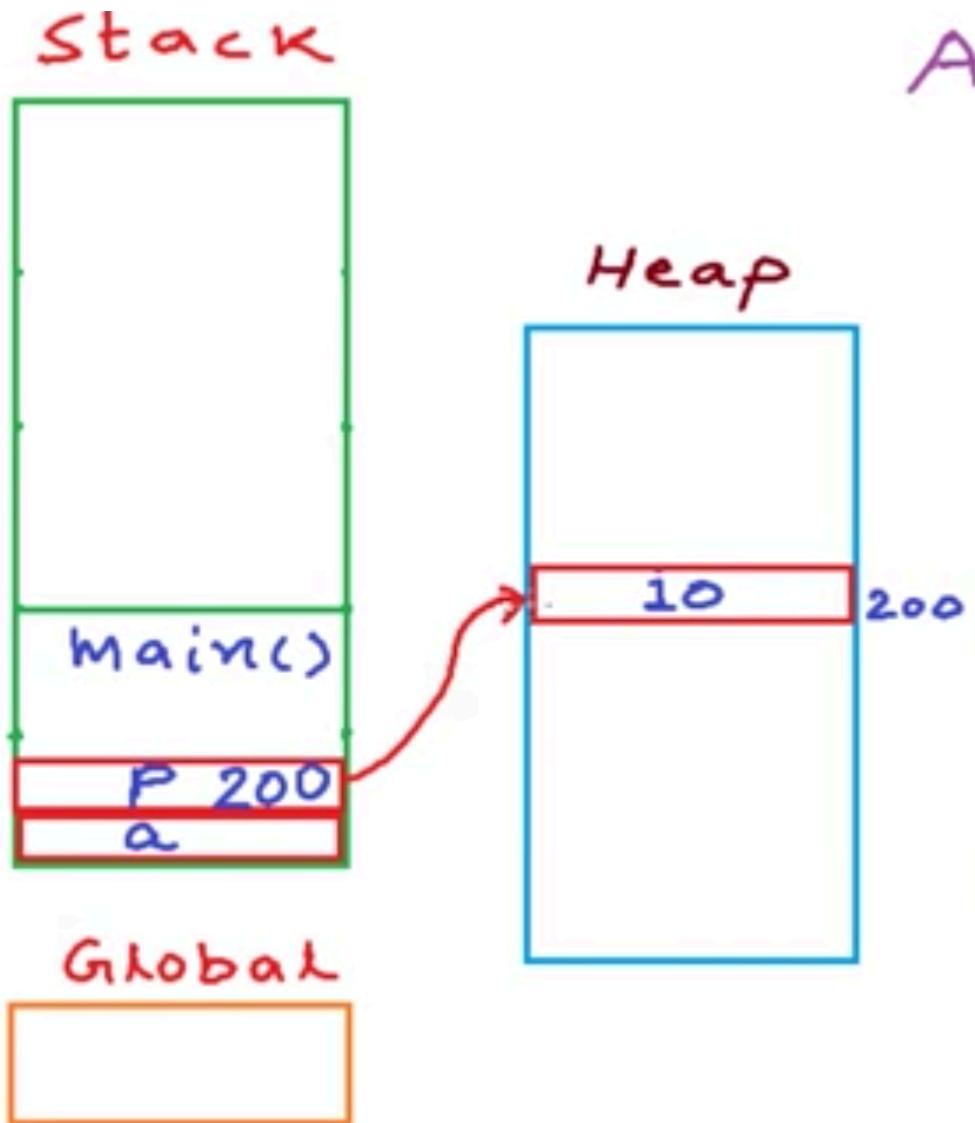
Demo: Stack vs Heap

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a; // goes on stack
    int *p;
    p = (int*)malloc(sizeof(int));
```



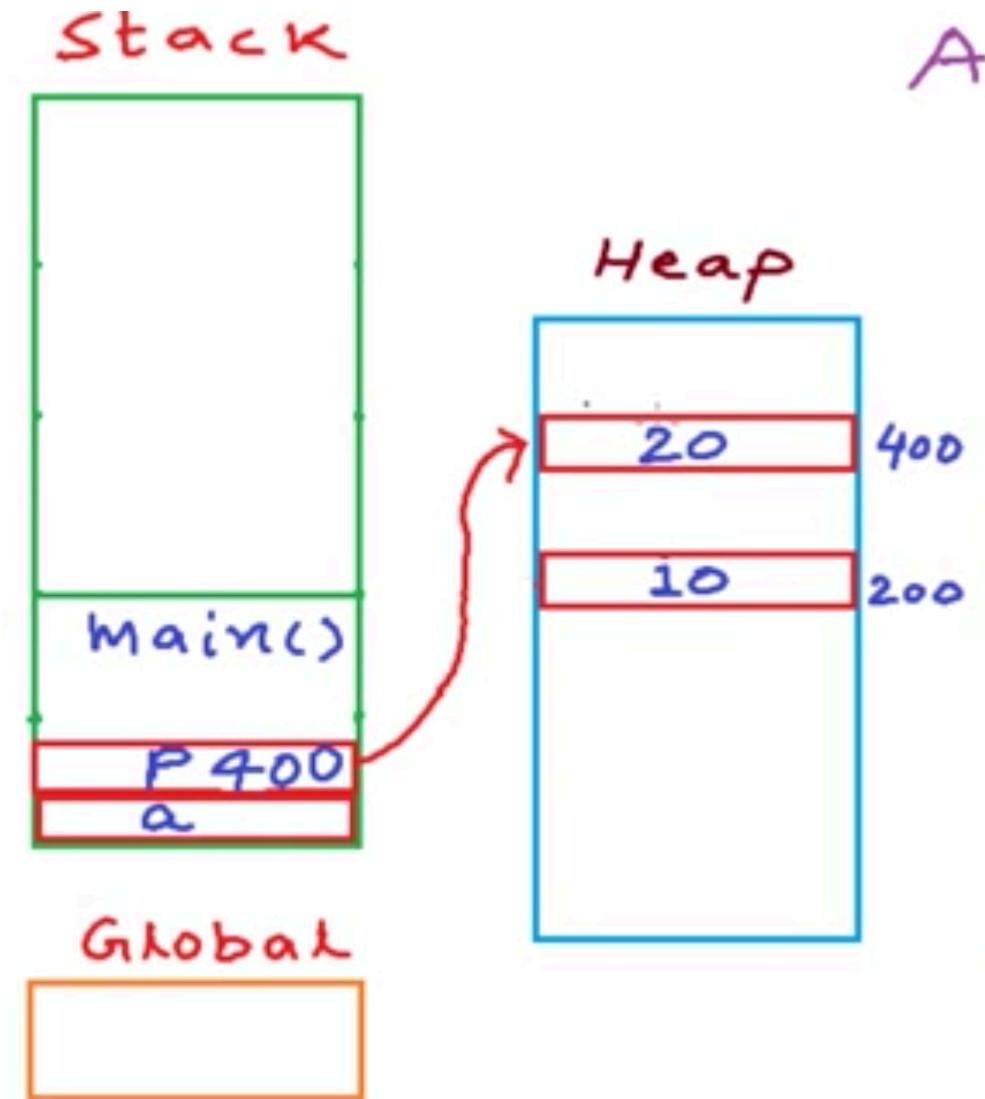
Dynamic memory allocation

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a; // goes on stack
    int *p;
    p = (int*)malloc(sizeof(int));
    *p = 10;
```



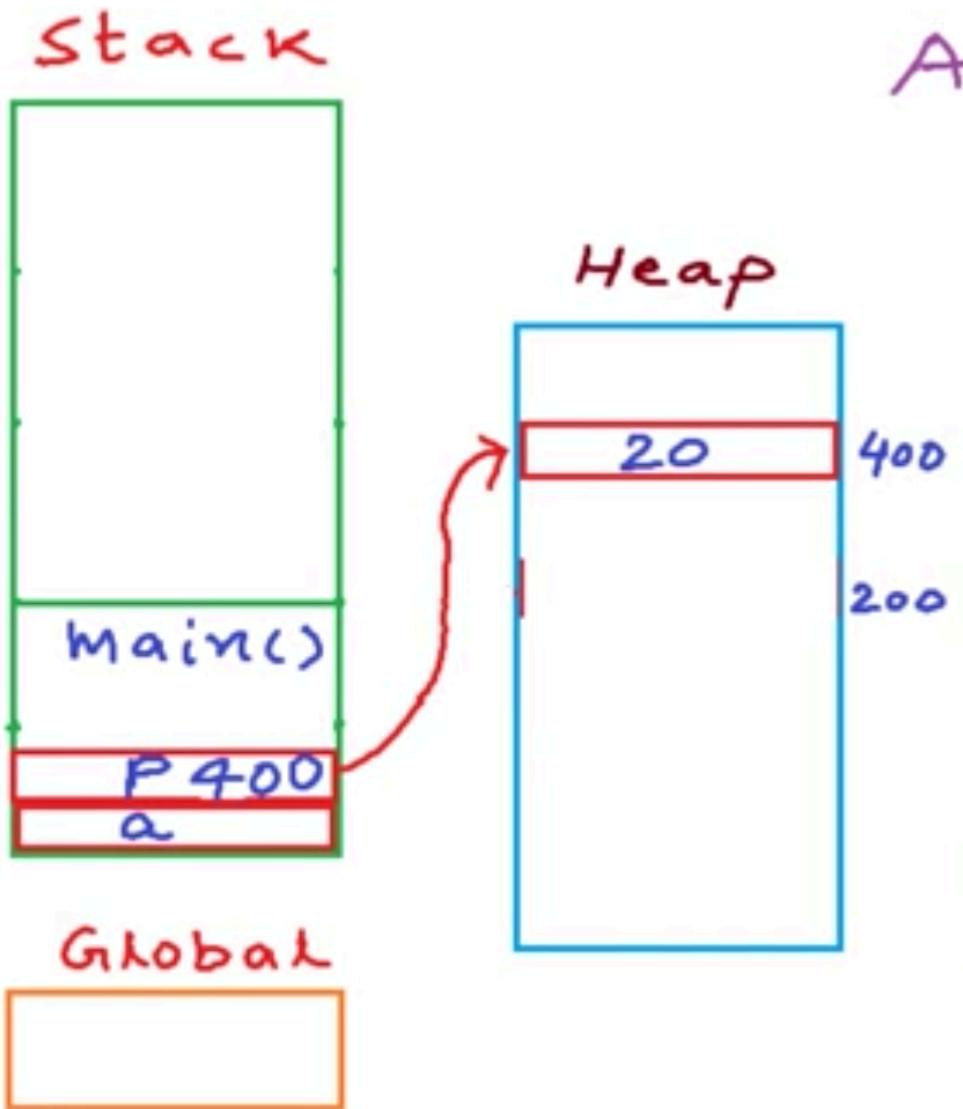
Dynamic memory allocation

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a; // goes on stack
    int *p;
    p = (int*)malloc(sizeof(int));
    *p = 10;
    p = (int*)malloc(sizeof(int));
    *p = 20;
```



Dynamic memory allocation

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a; // goes on stack
    int *p;
    p = (int*)malloc(sizeof(int));
    *p = 10;
    free(p);
    p = (int*)malloc(sizeof(int));
    *p = 20;
}
```



Reading text information

- Common streams and their file pointers
 - Standard input – stdin
 - Standard output – stdout
 - Standard error - stderr
- Reading/Writing characters
 - `getc(FILE *fp);`
 - `putc(int c, FILE *fp);`
- Reading/Writing Lines
 - `char *fgets(char *buf, int n, FILE *fp);`
 - `int fputs(const char *s, FILE *fp);`

Formatted Input/Output

- Open and close files
 - `FILE * fopen(const char * restrict filename, const char * restrict mode);`
 - `int fclose(FILE *fp);`
 - `FILE *fp` can be either a file pointer or `stdin`, `stdout`, or `stderr`
 - Output: `int fprintf(FILE * fp, const char * format, ...);`
 - Input: `int fscanf(FILE * fp, const char * format, ...);`
 - The format string:
 - `int score = 120; char player[] = "Mary";`
 - `printf("%s has %d points.\n", player, score);`
- Note: **restrict** says that the pointer is the only thing that accesses the underlying object. It eliminates the potential for pointer aliasing, enabling better optimization by the compiler.

Ternary Operator ?:

- Short form for a conditional assignment

`result = a > b ? x : y;` is equivalent to:

```
if(a>b)
{
    result = x;
}
else
{
    result = y;
}
```

Lab 4: get started

- Recall what you did in Lab 3
- Download the bugged version of coreutils
- Follow instructions in INSTALL (Similar steps in Assignment 3)
- Patching
 - There is an error when running make command
 - Download and edit the patch
 - Apply the patch using the patch command

Lab 4: Debugging and patching

- Test the files and reproduce the bug
- Use gdb to figure out what's wrong (running from the directory where the compiled ls lives)
- Construct a patch and fix the bug
 - Construct a patch containing your fix in a patch file called lab4.diff
 - It should contain a ChangeLog entry followed by the output of diff -u
- **Method to construct a patch file**
 - Modify the buggy file manually
 - Generate the patch: *diff -u* + redirection
 - Copy from /doc/ChangeLog and paste to your .diff file

Lab 4: Hints

- Try to reproduce the problem in your home directory, instead of the \$tmp directory
 - SEASnet NFS file system has 32-bit time stamps
 - Local File System on Linux server has 64-bit time stamps
 - If you touch the files on the NFS file system it will return timestamp around the year 2054
 - files have to be touched on local file system (**df -l**)
-