CS246
Unix: customization
C:function pointers

April 8

A Program that breaks

- gdb loves line numbers
  - cat -n xxx.c
- Program has three issues

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  void smashing() {
 5      int aa[10];
 6      for (int i = 0; i < 20; i++) {
 7          aa[i] = i;
 8      }
 9  }
10  int main(int argc, char const *argv[])
11  {
12      int strt = atoi(argv[1]);
13      int aa[strt];
14      smashing();
15      for (int i = 0; i < 1000; i++)
16      {
17          printf("%d %d\n", i, aa[i]);
18      }
19      return 0;
20  }
```

Customizing your Unix environment

- 2 files hold customizations
    - .bash_profile
        - executed when starting “login shells”
            - e.g. ssh
    - .bashrc
        - executed when starting non login shells
            - e.g. open a terminal window when logged in on lab machines
- Why 2?
    - .bash_profile set up your default environment that is used by everything
    - .bashrc sets up environment for terminals — you might want different things

.bash_profile and .bashrc defaults

- Ubuntu puts defaults for both in /etc/skel (??)

```
gtowell@benz:~$ cat /etc/skel/.profile
#umask 022
# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
            . "$HOME/.bashrc"
    fi
fi
# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/.local/bin" ] ; then
    PATH="$HOME/.local/bin:$PATH"
fi
```

.bash_profile — mine

- execute ==> UNIX> source .bash_profile
    - this is always done so you only need to do this if you edit and want to test

- Default just causes .bashrc to be executed

```
file: .bash_profile


if [ -f ~/.bashrc ]
then
   source ~/.bashrc
fi
```

exists and is a file

.bashrc — mine

- alias — similar to symbolic links but for commands
- export XX=YY
  - XX=YY is exactly setting a var as in shell scripts
  - shell scripts run in their own space so vars all die at end of script
  - export says make this var part of the shell in which you ran the command

```
alias lsls='ls -l | sort -k 5 -n -r'  # equivalent to ls -lS
export PATH=.:/home/gtowell/bin:/bin:$PATH:/usr/bin
alias coffee='echo "I need caffiene" '
alias cs_cd206='cd /www/htdocs/Courses/cs206/fall2020'
alias cs_cd380='cd /www/htdocs/Courses/cs380/fall2020'
alias more='less'

alias ls='ls --color=auto'
alias grep='grep —color=auto'

export PS1="\e[0;36m[\u@\h \W]> \e[0m”
```

More Customization

- the UNIX prompt:
  - `export PS1="\e[0;36m[\u@\h \w]> \e[0m"`
  - https://phoenixnap.com/kb/change-bash-prompt-linux

GDB — again

- Some more useful commands
    - Breakpoints
        - break file:linenumber
        - break functionName
        - info break
        - delete 2 # delete breakpoint 2
    - tui enable

- show on union2.c
    - points
        - backtrace
        - can still print and therefore see values of vars at time of death
        - set a breakpoint at line 68. Maybe conditional on loc>90

Function pointers

- Idea, a function has an address just like every other global variable. So, you can refer to a function by its address.

- Problem: WHAT IS THE TYPE OF A FUNCTION?
    - when working with pointers to normal variables you just say its type.

Function Pointers — calling

- Type is its signature — its prototype.
- That is, the type is almost exactly what you put in a .h file.

- Simplest usage
    - go to the thing pointed to by the name … give it the args …
    - The parens around (*mult) are required

```c
#include <stdio.h>

int mult(int a, int b) {
    return a * b;
}

int main(int argc, char const *argv[])
{
    printf("%d\n", (*mult)(3,4));
    return 0;
}
```

function pointers — passing

```
int(*f)(int,int)
```

return type

pointer to the function — parens are required

argument list, just the types, no names

Writing this with no spaces — emphasizes that it is one thing

```
int op(int(*f)(int,int), int a, int b) {
    return (*f)(a, b);
}
int mult(int a, int b) {
    return a * b;
}
int div(int a, int b) {
    return  (a / b)  + ((a % b == 0) ? 0 : 1);
}
int main(int argc, char const *argv[])
{
    printf("%d\n", op((*mult),3,4));
    printf("%d\n", op((*div),20,4));
    return 0;
}
```

function pointers sorting

- One of the major uses of function pointers across almost all PLs.
- define a generic sort function
    - take a comparison function as an argument
- Java — there are at least 3 ways to to this
    - comparitors are probably closest to C-like

```
public class Student {
    private String name;
    private int age;
    public Student(String n, int a) {
        name = n;
        age = a;
    }
    public String getName() { return name; }
    public int getAge() { return age; }
    public String toString() { return name + "-" + age;

    public static void main(String[] args) {
        Comparator<Student> cc = new Comparator<>() {
            public int compare(Student f, Student s) {
                return f.getAge() - s.getAge();
            }
        };
        Student[] ss = { new Student("J", 4), new Stude
        Arrays.sort(ss, cc);
        System.out.println(Arrays.toString(ss));
    }
}
```

function pointers — sorting

- C — the qsort function
  - stdlib.h
    - .h files ony contain prototypes
    - where is the function implementation???
- What does qsort need so much more than Arrays.sort??

The function used to compare elements
The Comparitor function

The number of elements in the array

The start of the array to be sorted

The size of each element in the array

```
void qsort(void *base, size_t nmemb, size_t size,
                int (*compar)(const void *, const void *));
```

Printing a la qsort
Stumbling towards generic-ness

- Works, but for one line in parray2

```
void parray2(void * base, size_t nmemb, size_t
size, char* fmt) {
    for (int i = 0; i < nmemb; i++) {
        char c = *((char *)base);
        printf(fmt, c);
        base += size;
    }
    printf("\n");
}

int main(int argc, char const *argv[]) {
    srand(time(NULL));
    char aa[100];
    for (int i = 0; i < 100; i++)
        aa[i] = 'a' + (rand() % 26);
    parray2(aa, 100, sizeof(char), "%c");
```

The problematic line. Why is this not simply a cast to char?

Comparison function for qsort

- 2 void pointers
  - at two items in array
- first thing, cast them to what you actually have in the array
- Then compare
  - In this case the comparison is trivial

```
int compareChar(const void * p, const void * q) {
    const char *pc = p;
    const char *qc = q;
    return *pc - *qc;
}
```

qsort
arrays of struct*

- Suppose a struct
    - standard constructor, destructor
- Have an array
    - of pointers to structs
- Want to use qsort

```c
typedef struct {
    char c;
    int n;
} ncs;
ncs* ncsMake() {
    ncs *ncsp = malloc(1 * sizeof(ncs));
    ncsp->c = 'A' + rand() % 26;
    ncsp->n = rand() % 100;
    return ncsp;
}
void printr(ncs** ncspp) {
    for (int i = 0; i < SIZ; i++) {
        printf("<%c %2d> ", ncspp[i]->c, ncspp[i]->n
    } printf("\n");
}
int main(int argc, char const *argv[])
{
    srand(time(NULL));
    ncs **ncspp = malloc(SIZ * sizeof(ncs *));
    for (int i = 0; i < SIZ; i++)
        ncspp[i] = ncsMake();
    printr(ncspp);
```

Comparison function

- Recall that array contains pointers to struct
    - Therefore the cast from void* must be to struct**
    - Then dereference once to get pointer to struct.
    - Then do comparison.
- Why const?
- What about the number in the struct?

```
int ncsCompareC(const void * p, const
void * q) {
    const ncs **ppp = (const ncs **) p;
    const ncs **qpp = (const ncs **) q;
    const ncs *pp = *ppp;
    const ncs *qp = *qpp;
    return pp->c - qp->c;
}
```

More comparisons

- Best thing to do is write a separate comparison function.
    - Goal, keep this function as simple as possible!!

```
int ncsCompareN(const void * p, const void * q) {
    const ncs **ppp = (const ncs **) p;
    const ncs **qpp = (const ncs **) q;
    const ncs *pp = *ppp;
    const ncs *qp = *qpp;
    return pp->n - qp->n;
}
```

More with function pointers

- suppose a file
    - op int int
- where “op” is a single character indicating an operation to perform on the two ints
    - simple approach
        - BIG switch with each op implemented inside the switch

- Use function pointers??!!

function pointers in arrays

- use typedef to make things easier
    - where is the name of the new type?
- Otherwise same as earlier example

```
typedef int(*myfuns)(int, int);

int op(int(*f)(int,int), int a, int b) {
    return (*f)(a, b);
}

int mult(int a, int b) {
    return a * b;
}

int divU(int a, int b) {
    return  (a / b)  + ((a % b == 0) ? 0 : 1);
}

int divV(int a, int b) {
    return  (a / b);
}
```

arrays of function pointers

```
int main(int argc, char const *argv[])
{
    myfuns funarr[128];
    funarr['m'] = mult;
    funarr['*'] = mult;
    funarr['d'] = divV;
    funarr['u'] = divU;

    char lin[256];
    char op;
    int v1;
    int v2;
    while (fgets(lin, 255, stdin)) {
        sscanf(lin, "%c %d %d", &op, &v1, &v2);
        if (op=='q')
            break;
        printf("%d\n", (funarr[op])(v1, v2));
    }
    return 0;
}
```

Legal!!!!

Danger here

Why??

:-(

Use fp

Compile then run within gdb

Lab

- suggest, and provide an implementation for, a way to make the array printer program truly generic
- Hint – function pointers!!!

```
file:  parray.c

void parray2(void * base, size_t nmemb, size_t size, char* fmt) {
    for (int i = 0; i < nmemb; i++) {
        char c = *((char *)base); // problematic line
        printf(fmt, c);
        base += size;
    }
    printf("\n");
}

int main(int argc, char const *argv[]) {
    srand(time(NULL));
    char aa[100];
    for (int i = 0; i < 100; i++)
        aa[i] = 'a' + (rand() % 26);
    parray2(aa, 100, sizeof(char), "%c");
}
```