CS246
Unix: shell script parameters
C:doing more with structs

April 21

Lab 4/19

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

file: printer.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");
}
```

A v3

- Problems ??

```
char * pprinter3(void * item) {
    char *citem = (char *)item;
    char *rtn = malloc(6 * sizeof(char));
    sprintf(rtn, "<%c>", *citem);
    return rtn;
}

void parray3(void * base, size_t nmemb, size_t size,
char *(*f)(void *)) {
    for (int i = 0; i < nmemb; i++) {
        printf("%s ", f(base));
        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);
    parray3(aa, 10, sizeof(char), pprinter3);
}
```

A v3a, v4

- free the memory!!
- Or statically declare char array and pass it
  - advantages/ disadvantages??

```
void parray3(void * base, size_t nmemb, size_t size,
char*(*f)(void*)) {
    for (int i = 0; i < nmemb; i++) {
        char *tmp = f(base);
        printf("%s ", tmp);
        base += size;
        free(tmp);
    }
    printf("\n");
}

void pprinter4(void * item, char* printInto) {
    char *citem = (char *)item;
    sprintf(printInto, "[%c]", *citem);
}

void parray4(void * base, size_t nmemb, size_t size,
void(*f)(void*,char*)) {
    char aa[100];
    for (int i = 0; i < nmemb; i++)
    {
        f(base, aa);
        printf("%s ", aa);
        base += size;
    }
    printf("\n");
}
```

Shell Scripts — command line params

- recall $1 … $9 are arguments 1..9 on command line
- $0 is the name of the script
  - just like main in C
- If you have more than 9, then ${12},
  - can do this on all … ${4} == $4
- BUT this does not
  - FOU=4
    ${$FOU}
- So … how to loop through command line args????

Working with lots of command line args

- When would this come up??
- Recall script from 2 weeks ago for totaling up size of executables with an extension
  - This was limited as it required that the files have a common extension.
  - Better would be just to take a set of files as command line args
    - for instance “p.sh *.exe L*”
      - This would pass to the script p.sh all files that either begin with L or end with .exe
        - There could be 100s

shell scripts with lots of command line params

- This approach works
  - kind of
- When there are less than 10 CLPs they get put into the string PP as blanks so the iteration only goes for existing values
- Problem: more than 10

```
#!/bin/bash

PP="$0 $1 $2 $3 $4 $5 $6 $7 $8 $9 ${10}“
for aa in $PP
do
    echo $aa
done
```

Handling larges number of CLPs

- $* and $@
  - both “expand into the list of positional parameters
- $#
  - the number of positional parameters
- GACKK — this is feeling like Perl

```
#!/bin/bash

for aa in $*
do
    echo "A $aa"
done

for aa in $@
do
    echo "B $aa"
done
```

Command line params

```
FILES=`ls *.${1}`
TOT=0
for FILE in $FILES
do
  DET=`ls -l $FILE`
  CNT=0
  for DETP in $DET
  do
     #echo "$CNT $DETP"
     if [ $CNT -eq 4 ]
     then
        TOT=$(( $TOT + $DETP ))
        #echo $DETP
     fi
     ((CNT++))
  done
done
echo $TOT
```

Added condition that item must be a file and be executable

file: summer.sh

Loop over all command line params

```
TOT=0
for FILE in $@
do
  if [[ -x $FILE  && -f $FILE ]]
  then
    DET=`ls -l $FILE`
    CNT=0
    for DETP in $DET
    do
      if [ $CNT -eq 4 ]
      then
        TOT=$(( $TOT + $DETP ))
        echo $FILE $DETP
      fi
    ((CNT++))
    done
  fi
done
echo $TOT
```

Shell Script Arrays

```
#!/bin/bash
AA=(ONE TWO THREE)
for ((i=0; i<${#AA[@]}; i++))
do
    echo $i ${AA[$i]} $AA[$i]
done
```

HAIRBALL!!!
actually the number of items in array

Create an array

Correct way to get array value

This will print the value of $AA followed by value of $i inside []

Putting this together

```
#!/bin/bash
TOT=0
for FILE in $@
do
  if [[ -x $FILE  && -f $FILE ]]
  then
    ADET=( `ls -l $FILE` )
    TOT=$(( $TOT + ${ADET[4]} ))
    echo $FILE ${ADET[4]}
    ((CNT++))
  fi
done
echo $TOT
```

String result of command is broken apart then put into array!

Using array index avoids awkward and strange loop.

Making structs more Object like

- transparent inheritance
- functions
- private vars

Transparent inheritance

- anonymous structures/ unions
- allow referring to items in included structures without annoying extra dots (which I always forget)
- MUST COMPILE USING gcc -fms-extensions

```
typedef struct {
    int x;
    int y;
} aa;
typedef struct {
   aa;
   int z;
} bb;
```

Simple use

```
#include <stdio.h>
typedef struct {
    int x;
    int y;
} aa;

typedef struct {
    union {
        aa;
        aa aaa;
    };
    int z;
} bb;

int main(int argc, char const *argv[])
{
    bb b;
    b.aaa.x = 10;
    b.y = 40;
    b.z = 20;
    printf("%d %d %d\n", b.z, b.aaa.x, b.y);
    return 0;
}
```

Anonymous union contains both anonymous struct and named struct.

You can use either!!

Using both!

function pointers in structs

- Why:
  - stylistic choice
  - can fewer naming collisions
    - the public name is inside struct so that name does not conflict with anything
  - In feels “Object-y”
- Why not:
  - space for the pointer
  - Cannot do this for constructors & destructors

```
file: sp1.h

typedef struct qq {
    int a;
    int b;
    void (*printer)(struct qq *);
} qq;
qq *makeQQ(int a, int b);
```

Name only exists within this file

```
file: sp1.c

#include <stdio.h>
#include "sp1.h"
#include <stdlib.h>
static void printer(qq * p) {
    printf("QQ: %d %d\n", p->a, p->b);
}
qq* makeQQ(int a, int b) {
    qq *q = malloc(1 * sizeof(qq));
    q->a = a;
    q->b = b;
    q->printer = printer;
}
```

function pointers in structs

- static keyword on printer function on previous page means that printer here does not conflict.
- ideally all “private” functions are static
- function is only “public” through pointer within the struct.

```
#include <stdio.h>
#include "sp1.h"

void printer(qq* q3) {
    printf("333 a:%d  b:%d\n", q3->a, q3->b);
}

int main(int argc, char const *argv[])
{
    qq *q = makeQQ(4, 5);
    q->printer(q);
    printer(q);
    return 0;
}
```

Private Vars

- DO NOT EXIST!!
- But
  - Privacy through naming
  - Idea: name variable to indicate they are private
  - provide well-named get/set
- People can still directly read but it is clear they should not.

```
typedef struct {
    int private1;
} privNAM;
int getYear(privNAM* pn) {
    return pn->private1;
}
void setYear(privNAM* pn, int yr) {
    pn->private1 = yr;
}
privNAM* makePrivNAM() {
    return malloc(1 * sizeof(privNAM));
}
int main(int argc, char const *argv[]) {
    privNAM *pn = makePrivNAM();
    setYear(pn, 1961);
    printf("%d\n", getYear(pn));
    return 0;
}
```

Private Variables

- Privacy through obscure indirection
  - Idea, store private vars behind void * pointers
  - again, well named get/set
- People can change the pointer, but clear they should not.
- Unreadable!! except through determined idiocy.

Only allocate space when needed

need to handle if not allocated

```
typedef struct {
    void* private1;
} privOI;
int getYear(privOI* pn) {
    int *ip = pn->private1;
    return *ip;
}
void setYear(privOI* pn, int yr) {
    if (pn->private1 == NULL) {
        pn->private1 = malloc(1 * sizeof(int));
    }
    int *ip = pn->private1;
    *ip = yr;
}
privOI* makePrivOI() {
    privOI* poi = malloc(1 * sizeof(privOI));
}
int main(int argc, char const *argv[]) {
    privOI *pn = makePrivOI();
    setYear(pn, 1961);
    printf("%d\n", getYear(pn));
    return 0;
}
```

Roll your own garbage collector

- Suppose a struct that contains string
  - its own private copy
- Also a substring struct
  - a pointer to the string struct
  - starting point for substring
  - length of substring
- Problem:
  - should not free the string struct until all of the substrings using it are free!
  - HOW??

- Create your own mini garbage collector!!

```
typedef struct {
    char *string;
} stringRC;

typedef struct {
    stringRC* string;
    int strt;
    int len;
} substringRC;

void printSubstr(substringRC* sstr) {
    for (int i = 0; i < sstr->len; i++
        printf("%c", sstr->string->str
    printf("\n");
}
void printString(stringRC *str) {
    printf("%s\n", str->string);
}
```

Implement reference counting

```
typedef struct {
    char *string;
    int refCount;
    int mainFreed;
} stringRC;

stringRC* makeStringRC(char * src) {
    // stuff not shown
    rc->refCount = 0;
    rc->mainFreed = 0;
    return rc;
}

substringRC*
makeSubstringRC(stringRC* str, int
st, int len) {
    // stuff not shown
    str->refCount++;
    return src;
}
```

Free main string iff there are no references to it
Otherwise mark it as ready to be freed

```
void freeStringRC(stringRC* str) {
    if (str->refCount==0) {
        printf("NO Refs -- freeing\n");
        free(str->string);
        free(str);
    } else {
        printf("Not freeing -- still has refs [%d
        str->mainFreed = 1;
    }
}

void freeSubstringRC(substringRC* src) {
    stringRC *tmp = src->string;
    free(src);
    tmp->refCount--;
    if (tmp->refCount==0 && tmp->mainFreed) {
        freeStringRC(tmp);
    }
}
```

call free on main string iff the only reason main string exists is because of references

```
int main(int argc, char const *argv[])
{
    stringRC *rc = makeStringRC("now
is the time of all good people to come
to the aid of their country");
    substringRC *src1 =
makeSubstringRC(rc, 0, 5);
    printSubstr(src1);
    freeSubstringRC(src1);
    printString(rc);
    freeStringRC(rc);
    return 0;
}
```

```
int main(int argc, char const *argv[])
{
    stringRC *rc = makeStringRC("now
is the time of all good people to come
to the aid of their country");
    substringRC *src1 =
makeSubstringRC(rc, 0, 5);
    printString(rc);
    freeStringRC(rc);
    printSubstr(src1);
    freeSubstringRC(src1);
   return 0;
}
```

Lab

- Write a shell script that sums the size of all writeable files in this directory and all directories immediately below this directory.
- The shell script should not do any directory traversal or listing on its own. Rather all of the file names should be given as command line parameters
- Send me a copy of your script and the unix command line through which this script would be invoked to do the above task.

- OR
  - today is earth day. Send me a picture of you outside, today. include something in the picture to show that the picture was taken today.
- OR
  - if you are on campus, go build/decorate a birdhouse. Send me a picture of your birdhouse