

C/C++ Refresher

ESE5320

Table of Contents

- Makefile
- Stack and Heap memory
- Pointers
- Common memory related mistakes in C

Makefile

target: dependencies

Action

`$@`: the filename representing the target

`$^`: the filenames of all the prerequisites

```
CXX = g++
CXXFLAGS = -std=c++11 -Wall -O3 -mcpu=native -fno-tree-vectorize -DNDEBUG -pthread
LDFLAGS = -lpthread -pthread
INCLUDES = -I./common

EXECUTABLES = baseline neon_filter example
SRC = $(wildcard src/*.cpp common/*.cpp)
EXAMPLE_SRC = $(wildcard neon_example/*.cpp common/*.cpp)

.PHONY: clean

all: $(EXECUTABLES)
    @echo $(EXECUTABLES) compiled!

baseline: $(SRC)
    $(CXX) $(CXXFLAGS) $(INCLUDES) $^ -o $@ $(LDFLAGS)

neon_filter: $(SRC)
    $(CXX) -DVECTORIZED $(CXXFLAGS) $(INCLUDES) $^ -o $@ $(LDFLAGS)

example: $(EXAMPLE_SRC)
    $(CXX) $(CXXFLAGS) $(INCLUDES) $^ -o $@ $(LDFLAGS)
```

<Example of Makefile>

Makefile

target: dependencies

Action

`$@`: the filename representing the target

`$^`: the filenames of all the prerequisites

```
CXX = g++
CXXFLAGS = -std=c++11 -Wall -O3 -mcpu=native -fno-tree-vectorize -DDEBUG -pthread
LDFLAGS = -lpthread -pthread
INCLUDES = -I./common

EXECUTABLES = baseline neon_filter example
SRC = $(wildcard src/*.cpp common/*.cpp)
EXAMPLE_SRC = $(wildcard neon_example/*.cpp common/*.cpp)

.PHONY: clean

all: $(EXECUTABLES)
    @echo $(EXECUTABLES) compiled!

baseline: $(SRC)
    $(CXX) $(CXXFLAGS) $(INCLUDES) $^ -o $@ $(LDFLAGS)

neon_filter: $(SRC)
    $(CXX) -DVECTORIZED $(CXXFLAGS) $(INCLUDES) $^ -o $@ $(LDFLAGS)

example: $(EXAMPLE_SRC)
    $(CXX) $(CXXFLAGS) $(INCLUDES) $^ -o $@ $(LDFLAGS)
```

<Example of Makefile>

```
(base) dopark@ubuntu:~/.../hw4/assignment$ make neon_filter
g++ -DVECTORIZED -std=c++11 -Wall -O3 -mcpu=native -fno-tree-vectorize -DDEBUG -pthread -I./common src/App.cpp src/Compress.cpp src/Differentiate.cpp src/Scale.cpp src/Filter.cpp common/Utilities.cpp -o neon_filter -lpthread -pthread
```

Makefile

VPP := v++

TARGET := hw

```
$(HOST): ${HOST_SRC}
    g++ ${GCC_OPTS} -I ${SOURCE_DIR} -o $$@ $$+
    @echo 'Compiled Host Executable: ${HOST_EXE}'

${XOS}: ${KERNEL_SRC}
    @${RM} $$@
    ${VPP} -t ${TARGET} --config ${CONFIG_DIR}/design.cfg \
        --log_dir ${LOG_DIR} \
        --report_dir ${REPORT_DIR} \
        --platform ${AWS_PLATFORM} \
        --compile --kernel ${NAME} \
        -D ${N} \
        -I ${SOURCE_DIR} -o $$@ $$+
    mv ${BUILD_DIR}/*compile_summary ${REPORT_DIR}/${NAME}.${TARGET}/

${XCLBIN}: ${XOS}
    ${VPP} -t ${TARGET} --config ${CONFIG_DIR}/design.cfg \
        --log_dir ${LOG_DIR} \
        --report_dir ${REPORT_DIR} \
        --platform ${AWS_PLATFORM} \
        --link -o $$@ $$+
    mv ${BUILD_DIR}/*link_summary ${REPORT_DIR}/${NAME}.${TARGET}/

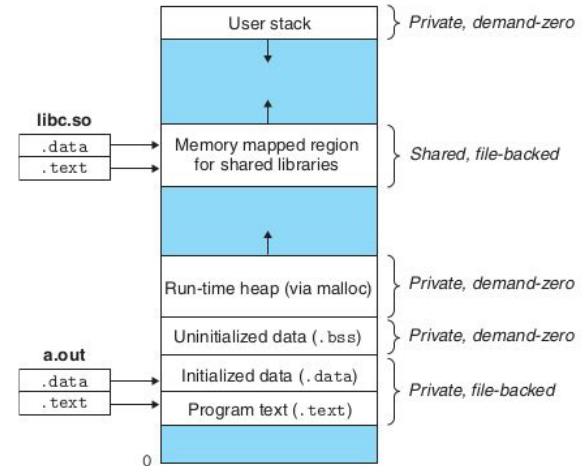
${EMCONFIG_FILE}:
    ${EMCONFIGUTIL} --platform ${AWS_PLATFORM} --od ${BUILD_DIR}

emulate: ${HOST} ${XCLBIN} ${EMCONFIG_FILE}
    echo Running host code with kernel...
    XCL_EMULATION_MODE=${TARGET} ./${HOST} ${XCLBIN}
    echo Finished run
```

<Example of Makefile used in Vitis flow>

Stack and Heap memory

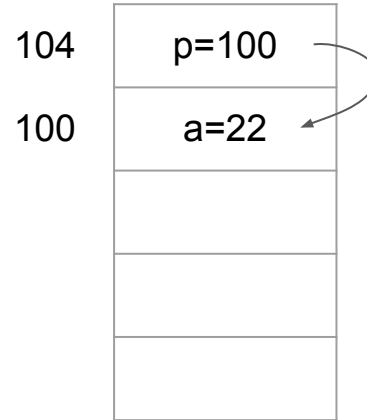
- stack: stores local variables
- heap: dynamic memory for programmer to allocate
- data: stores global variables, separated into initialized and uninitialized
- text: stores the code being executed



<User address space>*

Pointers

What are the print outputs?

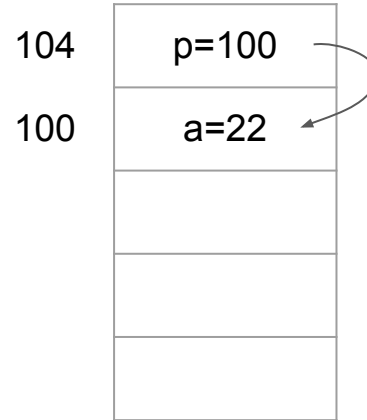


```
3 int main(){
4     int a;
5     int *p;
6     a = 22;
7     p = &a;
8     printf("p = %x\n", p);
9     printf("&a = %x\n", &a);
10    printf("&p = %x\n", &p);
11    printf("*p = %x\n", *p);
12    printf("p+1 = %x\n", p+1);
}
```

<Pointers basics, code and memory map>

Pointers

Answer: $p = 100$

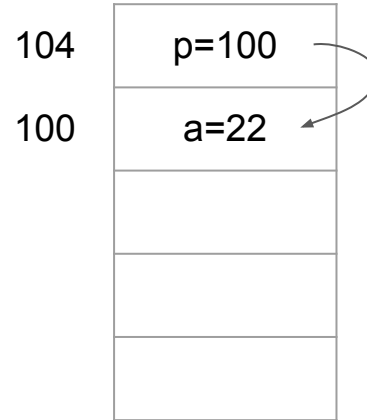


```
3 int main(){
4     int a;
5     int *p;
6     a = 22;
7     p = &a;
8     printf("p = %x\n", p);
9     printf("&a = %x\n", &a);
10    printf("&p = %x\n", &p);
11    printf("*p = %x\n", *p);
12    printf("p+1 = %x\n", p+1);
```

<Pointers basics, code and memory map>

Pointers

Answer: p = 100
 &a = 100

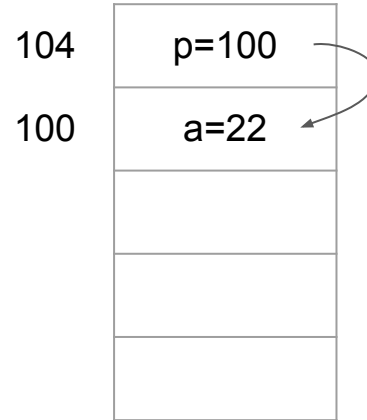


```
3  int main(){
4      int a;
5      int *p;
6      a = 22;
7      p = &a;
8      printf("p = %x\n", p);
9      printf("&a = %x\n", &a);
10     printf("&p = %x\n", &p);
11     printf("*p = %x\n", *p);
12     printf("p+1 = %x\n", p+1);
```

<Pointers basics, code and memory map>

Pointers

Answer: p = 100
 &a = 100
 &p = 104

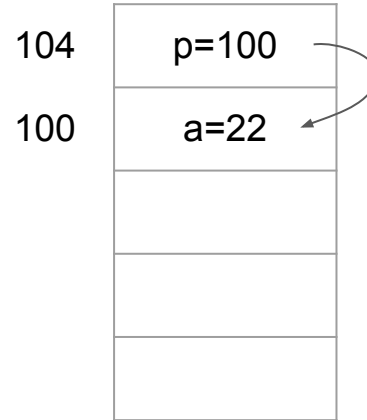


```
3  int main(){
4      int a;
5      int *p;
6      a = 22;
7      p = &a;
8      printf("p = %x\n", p);
9      printf("&a = %x\n", &a);
10     printf("&p = %x\n", &p);
11     printf("*p = %x\n", *p);
12     printf("p+1 = %x\n", p+1);
```

<Pointers basics, code and memory map>

Pointers

Answer: p = 100
 &a = 100
 &p = 104
 *p = 16 // hex



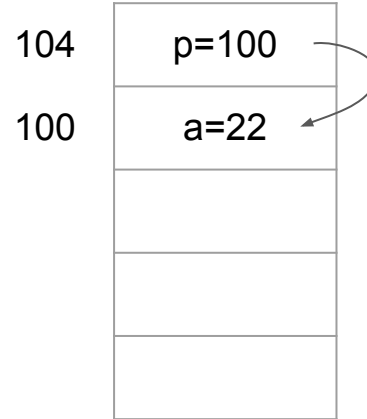
```
3 int main(){
4     int a;
5     int *p;
6     a = 22;
7     p = &a;
8     printf("p = %x\n", p);
9     printf("&a = %x\n", &a);
10    printf("&p = %x\n", &p);
11    printf("*p = %x\n", *p);
12    printf("p+1 = %x\n", p+1);
```

<Pointers basics, code and memory map>

Pointers

Answer:

```
p = 100  
&a = 100  
&p = 104  
*p = 16 // hex  
p+1 = 104
```



```
3 int main(){  
4     int a;  
5     int *p;  
6     a = 22;  
7     p = &a;  
8     printf("p = %x\n", p);  
9     printf("&a = %x\n", &a);  
10    printf("&p = %x\n", &p);  
11    printf("*p = %x\n", *p);  
12    printf("p+1 = %x\n", p+1);
```

<Pointers basics, code and memory map>

Pointers

What are the print outputs?

```
3 int main(){
4     int a = 1025;
5     int *p_0;
6     p_0 = &a;
7
8     char *p_1;
9     p_1 = (char*)p_0; // typecasting
10
11     printf("size of int = %d\n", sizeof(int)); // 4
12     printf("size of char = %d\n", sizeof(char)); // 1
13     printf("address = %d, value = %d\n", p_1, *p_1);
14     printf("address = %d, value = %d\n", p_1+1, *(p_1+1));
15
16     // hint: 1025 = 00000000 00000000 00000100 00000001
```

<Pointers typecasting, code>

Pointers

Answer:

address = 100, value = 1

address = 101, value = 4

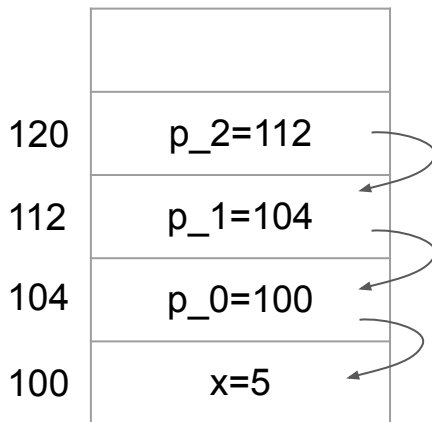
```
3 int main(){
4     int a = 1025;
5     int *p_0;
6     p_0 = &a;
7
8     char *p_1;
9     p_1 = (char*)p_0; // typecasting
10
11     printf("size of int = %d\n", sizeof(int)); // 4
12     printf("size of char = %d\n", sizeof(char)); // 1
13     printf("address = %d, value = %d\n", p_1, *p_1);
14     printf("address = %d, value = %d\n", p_1+1, *(p_1+1));
15
16     // hint: 1025 = 00000000 00000000 00000100 00000001
```

<Pointers typecasting, code>

Pointers

What are the print outputs?

Which line outputs 5?



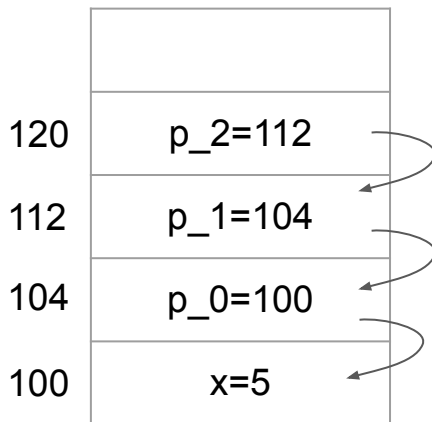
```
3 int main(){
4     int x = 5;
5     int *p_0;
6     int **p_1;
7     int ***p_2;
8
9     p_0 = &x;
10    p_1 = &p_0;
11    p_2 = &p_1;
12
13    printf("&x = %d\n", &x);
14    printf("&p_0 = %d\n", &p_0);
15    printf("&p_1 = %d\n", &p_1);
16    printf("&p_2 = %d\n", &p_2);
17
18    printf("*p_0 = %d\n", *p_0);
19    printf("*p_1 = %d\n", *p_1);
20    printf("**p_1 = %d\n", **p_1);
21    printf("*p_2 = %d\n", *p_2);
22    printf("***p_2 = %d\n", ***p_2);
23    printf("***p_2 = %d\n", ***p_2);
```

<Pointers to pointers, code and memory map>

Pointers

Answer:

&x = 100
&p_0 = 104
&p_1 = 112
&p_2 = 120



```
3 int main(){
4     int x = 5;
5     int *p_0;
6     int **p_1;
7     int ***p_2;
8
9     p_0 = &x;
10    p_1 = &p_0;
11    p_2 = &p_1;
12
13    printf("&x = %d\n", &x);
14    printf("&p_0 = %d\n", &p_0);
15    printf("&p_1 = %d\n", &p_1);
16    printf("&p_2 = %d\n", &p_2);
17
18    printf("*p_0 = %d\n", *p_0);
19    printf("*p_1 = %d\n", *p_1);
20    printf("**p_1 = %d\n", **p_1);
21    printf("*p_2 = %d\n", *p_2);
22    printf("***p_2 = %d\n", ***p_2);
23    printf("***p_2 = %d\n", ***p_2);
```

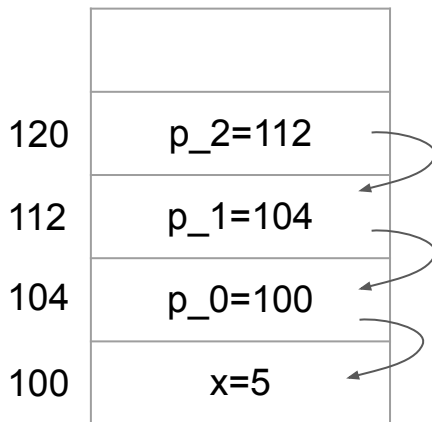
<Pointers to pointers, code and memory map>

Pointers

Answer:

```
&x    = 100
&p_0  = 104
&p_1  = 112
&p_2  = 120

*p_0   = 5
*p_1   = 100
**p_1  = 5
```



```
3 int main(){
4     int x = 5;
5     int *p_0;
6     int **p_1;
7     int ***p_2;
8
9     p_0 = &x;
10    p_1 = &p_0;
11    p_2 = &p_1;
12
13    printf("&x = %d\n", &x);
14    printf("&p_0 = %d\n", &p_0);
15    printf("&p_1 = %d\n", &p_1);
16    printf("&p_2 = %d\n", &p_2);
17
18    printf("*p_0 = %d\n", *p_0);
19    printf("*p_1 = %d\n", *p_1);
20    printf("**p_1 = %d\n", **p_1);
21    printf("*p_2 = %d\n", *p_2);
22    printf("***p_2 = %d\n", ***p_2);
23    printf("***p_2 = %d\n", ***p_2);
```

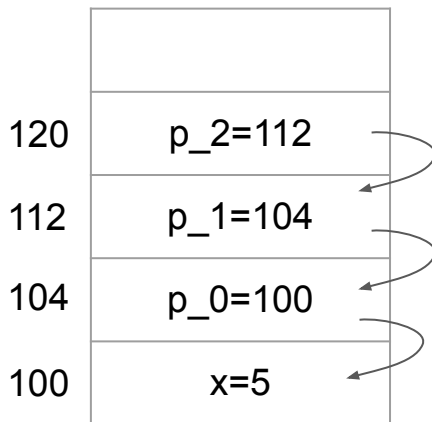
<Pointers to pointers, code and memory map>

Pointers

Answer:

```
&x    = 100
&p_0   = 104
&p_1   = 112
&p_2   = 120

*p_0    = 5
*p_1    = 100
**p_1   = 5
*p_2    = 104
**p_2   = 100
***p_2  = 5
```



```
3 int main(){
4     int x = 5;
5     int *p_0;
6     int **p_1;
7     int ***p_2;
8
9     p_0 = &x;
10    p_1 = &p_0;
11    p_2 = &p_1;
12
13    printf("&x = %d\n", &x);
14    printf("&p_0 = %d\n", &p_0);
15    printf("&p_1 = %d\n", &p_1);
16    printf("&p_2 = %d\n", &p_2);
17
18    printf("*p_0 = %d\n", *p_0);
19    printf("*p_1 = %d\n", *p_1);
20    printf("**p_1 = %d\n", **p_1);
21    printf("*p_2 = %d\n", *p_2);
22    printf("**p_2 = %d\n", **p_2);
23    printf("***p_2 = %d\n", ***p_2);
```

<Pointers to pointers, code and memory map>

Pointers

What are the print outputs?

116	A[4] = 5
112	A[3] = 4
108	A[2] = 3
104	A[1] = 2
100	A[0] = 1

```
3 int main(){
4     int A[] = {1,2,3,4,5};
5
6     printf("A = %d\n", A);
7     printf("&A = %d\n", &A);
8     printf("&A[0] = %d\n", &A[0]);
9     printf("A[0] = %d\n", A[0]);
10    printf("*A = %d\n", *A);
11    printf("A+1 = %d\n", A+1);
12    printf("A[1] = %d\n", A[1]);
13    printf("*(A+1) = %d\n", *(A+1));
```

<Pointers and arrays, code and memory map>

Pointers

Answer:

A = 100
&A = 100
&A[0] = 100

116	A[4] = 5
112	A[3] = 4
108	A[2] = 3
104	A[1] = 2
100	A[0] = 1

```
3 int main(){
4     int A[] = {1,2,3,4,5};
5
6     printf("A = %d\n", A);
7     printf("&A = %d\n", &A);
8     printf("&A[0] = %d\n", &A[0]);
9     printf("A[0] = %d\n", A[0]);
10    printf("*A = %d\n", *A);
11    printf("A+1 = %d\n", A+1);
12    printf("A[1] = %d\n", A[1]);
13    printf("*(A+1) = %d\n", *(A+1));
```

<Pointers and arrays, code and memory map>

Pointers

Answer:

A = 100
&A = 100
&A[0] = 100
A[0] = 1
*A = 1
A+1 = 104
A[1] = 2
*(A+1) = 2

116	A[4] = 5
112	A[3] = 4
108	A[2] = 3
104	A[1] = 2
100	A[0] = 1

```
3 int main(){
4     int A[] = {1,2,3,4,5};
5
6     printf("A = %d\n", A);
7     printf("&A = %d\n", &A);
8     printf("&A[0] = %d\n", &A[0]);
9     printf("A[0] = %d\n", A[0]);
10    printf("*A = %d\n", *A);
11    printf("A+1 = %d\n", A+1);
12    printf("A[1] = %d\n", A[1]);
13    printf("*(A+1) = %d\n", *(A+1));
```

<Pointers and arrays, code and memory map>

Pointers

What are the print outputs?

120

A[1][2] = 6

116

A[1][1] = 5

112

A[1][0] = 4

108

A[0][2] = 3

104

A[0][1] = 2

100

A[0][0] = 1

```
3 int main(){
4     int A[2][3] = {{1,2,3},
5                     {4,5,6}};
6
7     printf("A = %d\n", A);
8     printf("&A = %d\n", &A);
9     printf("&A[0] = %d\n", &A[0]);
10
11     printf("A[0] = %d\n", A[0]);
12     printf("&A[0][0] = %d\n", &A[0][0]);
13     printf("*A = %d\n", *A);
14
15     printf("A[0][0] = %d\n", A[0][0]);
16
17     printf("A+1 = %d\n", A+1);
18     printf("A[1] = %d\n", A[1]);
19     printf("*(A+1) = %d\n", *(A+1));
20
21     printf("*(A+1)+2 = %d\n", *(A+1)+2);
22     printf("A[1]+2 = %d\n", A[1]+2);
23     printf("&A[1][2] = %d\n", &A[1][2]);
24
25     printf("*(*(A+1)+2) = %d\n", (*(A+1)+2));
26     printf("*A[1]+2 = %d\n", *A[1]+2);
27     printf("A[1][2] = %d\n", A[1][2]);
```

<Pointers and 2D arrays, code and memory map>

Pointers

Answer:

A = 100
&A = 100
&A[0] = 100

A[1]	120	A[1][2] = 6
	116	A[1][1] = 5
	112	A[1][0] = 4
A[0]	108	A[0][2] = 3
	104	A[0][1] = 2
	100	A[0][0] = 1

```
3 int main(){
4     int A[2][3] = {{1,2,3},
5                     {4,5,6}};
6
7     printf("A = %d\n", A);
8     printf("&A = %d\n", &A);
9     printf("&A[0] = %d\n", &A[0]);
10
11    printf("A[0] = %d\n", A[0]);
12    printf("&A[0][0] = %d\n", &A[0][0]);
13    printf("*A = %d\n", *A);
14
15    printf("A[0][0] = %d\n", A[0][0]);
16
17    printf("A+1 = %d\n", A+1);
18    printf("A[1] = %d\n", A[1]);
19    printf("*(A+1) = %d\n", *(A+1));
20
21    printf("*(A+1)+2 = %d\n", *(A+1)+2);
22    printf("A[1]+2 = %d\n", A[1]+2);
23    printf("&A[1][2] = %d\n", &A[1][2]);
24
25    printf("*(*(A+1)+2) = %d\n", (*(A+1)+2));
26    printf("*A[1]+2 = %d\n", *A[1]+2);
27    printf("A[1][2] = %d\n", A[1][2]);
```

<Pointers and 2D arrays, code and memory map>

Pointers

Answer:

A = 100
&A = 100
&A[0] = 100

A[0] = 100
&A[0][0] = 100
*A = 100

A[0][0] = 1

A[1]	120	A[1][2] = 6
	116	A[1][1] = 5
	112	A[1][0] = 4
A[0]	108	A[0][2] = 3
	104	A[0][1] = 2
	100	A[0][0] = 1

```
3 int main(){
4     int A[2][3] = {{1,2,3},
5                     {4,5,6}};
6
7     printf("A = %d\n", A);
8     printf("&A = %d\n", &A);
9     printf("&A[0] = %d\n", &A[0]);
10
11    printf("A[0] = %d\n", A[0]);
12    printf("&A[0][0] = %d\n", &A[0][0]);
13    printf("*A = %d\n", *A);
14
15    printf("A[0][0] = %d\n", A[0][0]);
16
17    printf("A+1 = %d\n", A+1);
18    printf("A[1] = %d\n", A[1]);
19    printf("*(A+1) = %d\n", *(A+1));
20
21    printf("*(A+1)+2 = %d\n", *(A+1)+2);
22    printf("A[1]+2 = %d\n", A[1]+2);
23    printf("&A[1][2] = %d\n", &A[1][2]);
24
25    printf("(*(A+1)+2) = %d\n", (*(A+1)+2));
26    printf("*(A[1]+2) = %d\n", *(A[1]+2));
27    printf("A[1][2] = %d\n", A[1][2]);
```

<Pointers and 2D arrays, code and memory map>

Pointers

Answer:

A = 100
&A = 100
&A[0] = 100

A[0] = 100
&A[0][0] = 100
*A = 100

A[0][0] = 1

A+1 = 112
A[1] = 112
*(A+1) = 112

A[1]	120	A[1][2] = 6
	116	A[1][1] = 5
	112	A[1][0] = 4
A[0]	108	A[0][2] = 3
	104	A[0][1] = 2
	100	A[0][0] = 1

```
3 int main(){
4     int A[2][3] = {{1,2,3},
5                     {4,5,6}};
6
7     printf("A = %d\n", A);
8     printf("&A = %d\n", &A);
9     printf("&A[0] = %d\n", &A[0]);
10
11    printf("A[0] = %d\n", A[0]);
12    printf("&A[0][0] = %d\n", &A[0][0]);
13    printf("*A = %d\n", *A);
14
15    printf("A[0][0] = %d\n", A[0][0]);
16
17    printf("A+1 = %d\n", A+1);
18    printf("A[1] = %d\n", A[1]);
19    printf("*(A+1) = %d\n", *(A+1));
20
21    printf("*(A+1)+2 = %d\n", *(A+1)+2);
22    printf("A[1]+2 = %d\n", A[1]+2);
23    printf("&A[1][2] = %d\n", &A[1][2]);
24
25    printf("(*(A+1)+2) = %d\n", (*(A+1)+2));
26    printf("*(A[1]+2) = %d\n", *(A[1]+2));
27    printf("A[1][2] = %d\n", A[1][2]);
```

<Pointers and 2D arrays, code and memory map>

Pointers

Answer:

A = 100
&A = 100
&A[0] = 100

A[0] = 100
&A[0][0] = 100
*A = 100

A[0][0] = 1

A+1 = 112
A[1] = 112
*(A+1) = 112

*(A+1)+2 = 120
A[1]+2 = 120
&A[1][2] = 120

A[1]	120	A[1][2] = 6
	116	A[1][1] = 5
	112	A[1][0] = 4
A[0]	108	A[0][2] = 3
	104	A[0][1] = 2
	100	A[0][0] = 1

```
3 int main(){
4     int A[2][3] = {{1,2,3},
5                     {4,5,6}};
6
7     printf("A = %d\n", A);
8     printf("&A = %d\n", &A);
9     printf("&A[0] = %d\n", &A[0]);
10
11     printf("A[0] = %d\n", A[0]);
12     printf("&A[0][0] = %d\n", &A[0][0]);
13     printf("*A = %d\n", *A);
14
15     printf("A[0][0] = %d\n", A[0][0]);
16
17     printf("A+1 = %d\n", A+1);
18     printf("A[1] = %d\n", A[1]);
19     printf("*(A+1) = %d\n", *(A+1));
20
21     printf("*(A+1)+2 = %d\n", *(A+1)+2);
22     printf("A[1]+2 = %d\n", A[1]+2);
23     printf("&A[1][2] = %d\n", &A[1][2]);
24
25     printf("*(*(A+1)+2) = %d\n", (*(A+1)+2));
26     printf("*A[1]+2 = %d\n", *A[1]+2);
27     printf("A[1][2] = %d\n", A[1][2]);
```

<Pointers and 2D arrays, code and memory map>

Pointers

Answer:

A = 100
&A = 100
&A[0] = 100

A[0] = 100
&A[0][0] = 100
*A = 100

A[0][0] = 1

A+1 = 112
A[1] = 112
*(A+1) = 112

*(A+1)+2 = 120
A[1]+2 = 120
&A[1][2] = 120

*A+1 = 6
*(A[1]+2) = 6
A[1][2] = 6

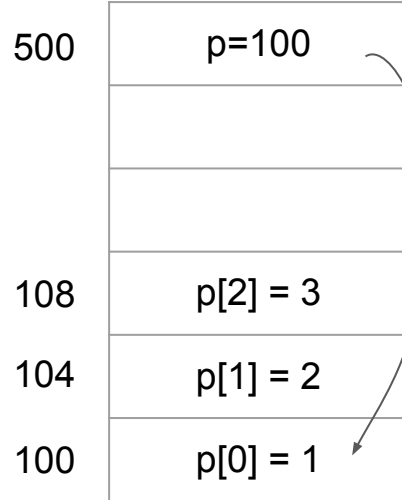
A[1]	120	A[1][2] = 6
	116	A[1][1] = 5
	112	A[1][0] = 4
A[0]	108	A[0][2] = 3
	104	A[0][1] = 2
	100	A[0][0] = 1

```
3 int main(){
4     int A[2][3] = {{1,2,3},
5                     {4,5,6}};
6
7     printf("A = %d\n", A);
8     printf("&A = %d\n", &A);
9     printf("&A[0] = %d\n", &A[0]);
10
11     printf("A[0] = %d\n", A[0]);
12     printf("&A[0][0] = %d\n", &A[0][0]);
13     printf("*A = %d\n", *A);
14
15     printf("A[0][0] = %d\n", A[0][0]);
16
17     printf("A+1 = %d\n", A+1);
18     printf("A[1] = %d\n", A[1]);
19     printf("*(A+1) = %d\n", *(A+1));
20
21     printf("*(A+1)+2 = %d\n", *(A+1)+2);
22     printf("A[1]+2 = %d\n", A[1]+2);
23     printf("&A[1][2] = %d\n", &A[1][2]);
24
25     printf("*(*(A+1)+2) = %d\n", (*(A+1)+2));
26     printf("*(A[1]+2) = %d\n", *(A[1]+2));
27     printf("A[1][2] = %d\n", A[1][2]);
```

<Pointers and 2D arrays, code and memory map>

Pointers

What are the print outputs?



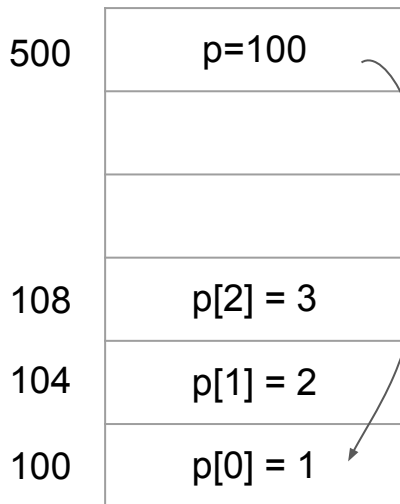
```
4 int main(){
5     int a;
6     int *p;
7     p = (int*)malloc(3*sizeof(int));
8     p[0] = 1;
9     p[1] = 2;
10    p[2] = 3;
11
12    printf("&p = %p\n", &p);
13    printf("p = %p\n", p);
14
15    printf("&p[0] = %p\n", &p[0]);
16    printf("&p[1] = %p\n", &p[1]);
17    printf("&p[2] = %p\n", &p[2]);
18
19    printf("p[0] = %d\n", p[0]);
20    printf("p[1] = %d\n", p[1]);
21    printf("p[2] = %d\n", p[2]);
22
23    free(p);
}
```

<Pointers and dynamic memory, code and memory map>

Pointers

Answer:

&p = 500 // stack
p = 100 // heap



```
4 int main(){
5     int a;
6     int *p;
7     p = (int*)malloc(3*sizeof(int));
8     p[0] = 1;
9     p[1] = 2;
10    p[2] = 3;
11
12    printf("&p = %p\n", &p);
13    printf("p = %p\n", p);
14
15    printf("&p[0] = %p\n", &p[0]);
16    printf("&p[1] = %p\n", &p[1]);
17    printf("&p[2] = %p\n", &p[2]);
18
19    printf("p[0] = %d\n", p[0]);
20    printf("p[1] = %d\n", p[1]);
21    printf("p[2] = %d\n", p[2]);
22
23    free(p);
}
```

<Pointers and dynamic memory, code and memory map>

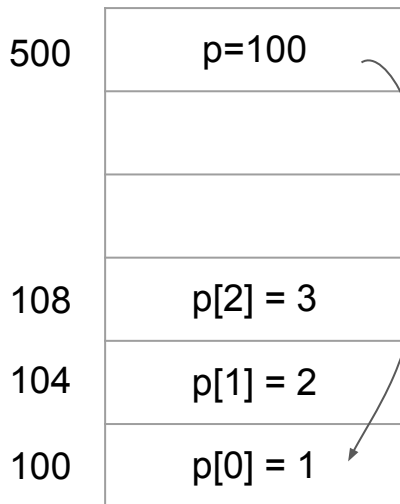
Pointers

Answer:

&p = 500 // stack
p = 100 // heap

&p[0] = 100 // heap
&p[1] = 104 // heap
&p[2] = 108 // heap

p[0] = 1
p[1] = 2
p[2] = 3



```
4 int main(){
5     int a;
6     int *p;
7     p = (int*)malloc(3*sizeof(int));
8     p[0] = 1;
9     p[1] = 2;
10    p[2] = 3;
11
12    printf("&p = %p\n", &p);
13    printf("p = %p\n", p);
14
15    printf("&p[0] = %p\n", &p[0]);
16    printf("&p[1] = %p\n", &p[1]);
17    printf("&p[2] = %p\n", &p[2]);
18
19    printf("p[0] = %d\n", p[0]);
20    printf("p[1] = %d\n", p[1]);
21    printf("p[2] = %d\n", p[2]);
22
23    free(p);
}
```

<Pointers and dynamic memory, code and memory map>

Common memory related mistakes in C

What is wrong?

```
1  /* Create an nxm array */
2  int **makeArray1(int n, int m)
3  {
4      int i;
5      int **A = (int **)Malloc(n * sizeof(int));
6
7      for (i = 0; i < n; i++)
8          A[i] = (int *)Malloc(m * sizeof(int));
9      return A;
10 }
```

<Assuming That Pointers and the Objects They Point to Are the Same Size>*

Common memory related mistakes in C

What is wrong?

```
1  /* Create an nxm array */
2  int **makeArray1(int n, int m)
3  {
4      int i;
5      int **A = (int **)Malloc(n * sizeof(int));
6
7      for (i = 0; i < n; i++)
8          A[i] = (int *)Malloc(m * sizeof(int));
9      return A;
10 }
```

`sizeof(int*)`

<Assuming That Pointers and the Objects They Point to Are the Same Size>*

Common memory related mistakes in C

What is wrong?

```
1  int *search(int *p, int val)
2  {
3      while (*p && *p != val)
4          p += sizeof(int); /* Should be p++ */
5      return p;
6  }
```

<Misunderstanding Pointer Arithmetic>*

References

- mycodeschool, Pointers in C/C++
- “Computer Systems, A Programmer’s Perspective”

Pointers (swap - 1)

What are the print outputs?

```
4 void swap1(int a, int b) {  
5     int temp = a;  
6     a = b;  
7     b = temp;  
8     assert(a == 17);  
9     assert(b == 42);  
10    // they're swapped!  
11 }  
12  
13  
14 int main(){  
15     int x = 42;  
16     int y = 17;  
17     swap1(x, y);  
18     printf("x=%d\n", x);  
19     printf("y=%d\n", y);
```

Pointers (swap - 1)

Answer: x = 42
 y = 17

```
4 void swap1(int a, int b) {  
5     int temp = a;  
6     a = b;  
7     b = temp;  
8     assert(a == 17);  
9     assert(b == 42);  
10    // they're swapped!  
11 }  
12  
13  
14 int main(){  
15     int x = 42;  
16     int y = 17;  
17     swap1(x, y);  
18     printf("x=%d\n", x);  
19     printf("y=%d\n", y);
```

Pointers (swap - 2)

What are the print outputs?

```
4 void swap2(int* a, int* b) {  
5     int temp = *a;  
6     *a = *b;  
7     *b = temp;  
8     assert(*a == 17);  
9     assert(*b == 42);  
10    // they're swapped!  
11 }  
12  
13  
14  
15 int main(){  
16     int x = 42;  
17     int y = 17;  
18     swap2(&x, &y);  
19     printf("x=%d\n", x);  
20     printf("y=%d\n", y);
```

Pointers (swap - 2)

Answer: x = 17
 y = 42

```
4 void swap2(int* a, int* b) {  
5     int temp = *a;  
6     *a = *b;  
7     *b = temp;  
8     assert(*a == 17);  
9     assert(*b == 42);  
10    // they're swapped!  
11 }  
12  
13  
14  
15 int main(){  
16     int x = 42;  
17     int y = 17;  
18     swap2(&x, &y);  
19     printf("x=%d\n", x);  
20     printf("y=%d\n", y);
```

Pointers (swap - 3)

What are the print outputs?

&x=0x100

&y=0x104

xp=?

yp=?

x=?

y=?

```
13 void swap4(int* a, int* b) {
14     int temp = *a;
15     *a = *b;
16     *b = temp;
17     assert(*a == 17);
18     assert(*b == 42);
19     // they're swapped!
20 }
21
22
23 int main(){
24
25     int x = 42;
26     int y = 17;
27     int* xp = &x;
28     int* yp = &y;
29
30     swap4(xp, yp);
31     printf("&x=%p\n", &x);
32     printf("&y=%p\n", &y);
33     printf("xp=%p\n", xp);
34     printf("yp=%p\n", yp);
35     printf("x=%d\n", x);
36     printf("y=%d\n", y);
```

Pointers (swap - 3)

Answer:

&x=0x100
&y=0x104
xp=0x100
yp=0x104
x=17
y=42

```
13 void swap4(int* a, int* b) {  
14     int temp = *a;  
15     *a = *b;  
16     *b = temp;  
17     assert(*a == 17);  
18     assert(*b == 42);  
19     // they're swapped!  
20 }  
21  
22  
23 int main(){  
24  
25     int x = 42;  
26     int y = 17;  
27     int* xp = &x;  
28     int* yp = &y;  
29  
30     swap4(xp, yp);  
31     printf("&x=%p\n", &x);  
32     printf("&y=%p\n", &y);  
33     printf("xp=%p\n", xp);  
34     printf("yp=%p\n", yp);  
35     printf("x=%d\n", x);  
36     printf("y=%d\n", y);
```


Pointers (swap - 5)

What are the print outputs?

&x=0x100

&y=0x104

xp=?

yp=?

x=?

y=?

```
4 void swap5(int** a, int** b) {  
5     int* temp = *a;  
6     *a = *b;  
7     *b = temp;  
8     assert(**a == 17);  
9     assert(**b == 42);  
10    // they're swapped!  
11 }  
12  
13  
14 int main(){  
15  
16     int x = 42;  
17     int y = 17;  
18     int* xp = &x;  
19     int* yp = &y;  
20  
21     swap5(&xp, &yp);  
22     printf("&x=%p\n", &x);  
23     printf("&y=%p\n", &y);  
24     printf("xp=%p\n", xp);  
25     printf("yp=%p\n", yp);  
26     printf("x=%d\n", x);  
27     printf("y=%d\n", y);
```

Pointers (swap - 5)

Answer:

&x=0x100
&y=0x104
xp=0x104
yp=0x100
x=42
y=17

```
4 void swap5(int** a, int** b) {  
5     int* temp = *a;  
6     *a = *b;  
7     *b = temp;  
8     assert(**a == 17);  
9     assert(**b == 42);  
10    // they're swapped!  
11 }  
12  
13  
14 int main(){  
15  
16     int x = 42;  
17     int y = 17;  
18     int* xp = &x;  
19     int* yp = &y;  
20  
21     swap5(&xp, &yp);  
22     printf("&x=%p\n", &x);  
23     printf("&y=%p\n", &y);  
24     printf("xp=%p\n", xp);  
25     printf("yp=%p\n", yp);  
26     printf("x=%d\n", x);  
27     printf("y=%d\n", y);
```