**Seminar 11**

**Котет Ахмед P3211**

**Question** compile the next file and see the contents of the section .data using objdump. At what address does the variable start x? Try to remove \_Alignas(128)and explain the effect.

Compile the file and run the command **objdump -D FILENAME.o**

With \_Alignas(128):

Изображение выглядит как текст

Автоматически созданное описание

Without \_Alignas(128):

Изображение выглядит как текст

Автоматически созданное описание

This command simply makes sure that any allocated data is allocated from addresses that are multiples of the argument, it means that you can allocate data in 10000 to 1 byte.

**Printer Program:**

#include <inttypes.h>

#include <malloc.h>

#include <stdio.h>

#include <string.h>

struct AST **{**

enum AST\_type **{** AST\_BINOP**,** AST\_UNOP**,** AST\_LIT **}** type**;**

union **{**

struct binop **{**

enum binop\_type **{** BIN\_PLUS**,** BIN\_MINUS**,** BIN\_MUL**,** BIN\_DIV **}** type**;**

struct AST **\***left**,** **\***right**;**

**}** as\_binop**;**

struct unop **{**

enum unop\_type **{** UN\_NEG **}** type**;**

struct AST **\***operand**;**

**}** as\_unop**;**

struct literal **{**

int64\_t value**;**

**}** as\_literal**;**

**};**

**};**

/\* DSL \*/

static struct AST **\***newnode**(**struct AST ast**)** **{**

struct AST **\***const node **=** malloc**(sizeof(**struct AST**));**

**\***node **=** ast**;**

**return** node**;**

**}**

struct AST \_lit**(**int64\_t value**)** **{**

**return** **(**struct AST**){**AST\_LIT**,** **.**as\_literal **=** **{**value**}};**

**}**

struct AST **\***lit**(**int64\_t value**)** **{**

**return** newnode**(**\_lit**(**value**));**

**}**

struct AST \_unop**(**enum unop\_type type**,** struct AST **\***operand**)** **{**

**return** **(**struct AST**){**AST\_UNOP**,** **.**as\_unop **=** **{**type**,** operand**}};**

**}**

struct AST **\***unop**(**enum unop\_type type**,** struct AST **\***operand**)** **{**

**return** newnode**(**\_unop**(**type**,** operand**));**

**}**

struct AST \_binop**(**enum binop\_type type**,** struct AST **\***left**,** struct AST **\***right**)** **{**

**return** **(**struct AST**){**AST\_BINOP**,** **.**as\_binop **=** **{**type**,** left**,** right**}};**

**}**

struct AST **\***binop**(**enum binop\_type type**,** struct AST **\***left**,** struct AST **\***right**)** **{**

**return** newnode**(**\_binop**(**type**,** left**,** right**));**

**}**

#define DECLARE\_BINOP(fun, code) \

struct AST \*fun(struct AST \*left, struct AST \*right) { \

return binop(BIN\_##code, left, right); \

}

DECLARE\_BINOP**(**add**,** PLUS**)**

DECLARE\_BINOP**(**mul**,** MUL**)**

DECLARE\_BINOP**(**sub**,** MINUS**)**

DECLARE\_BINOP**(**div**,** DIV**)**

#undef DECLARE\_BINOP

#define DECLARE\_UNOP(fun, code) \

struct AST \*fun(struct AST \*operand) { \

return unop(UN\_##code, operand); \

}

DECLARE\_UNOP**(**neg**,** NEG**)**

#undef DECLARE\_UNOP

/\* printer \*/

static const char **\***BINOPS**[]** **=** **{**

**[**BIN\_PLUS**]** **=** "+"**,** **[**BIN\_MINUS**]** **=** "-"**,** **[**BIN\_MUL**]** **=** "\*"**,** **[**BIN\_DIV**]** **=** "/"**};**

static const char **\***UNOPS**[]** **=** **{[**UN\_NEG**]** **=** "-"**};**

**typedef** void**(**printer**)(**FILE **\*,** struct AST **\*);**

void print**(**FILE **\***f**,** struct AST **\***ast**);**

void print\_binop**(**FILE **\***f**,** struct AST **\***ast**)** **{**

fprintf**(**f**,** "("**);**

print**(**f**,** ast**->**as\_binop**.**left**);**

fprintf**(**f**,** ")"**);**

fprintf**(**f**,** "%s"**,** BINOPS**[**ast**->**as\_binop**.**type**]);**

fprintf**(**f**,** "("**);**

print**(**f**,** ast**->**as\_binop**.**right**);**

fprintf**(**f**,** ")"**);**

**}**

void print\_unop**(**FILE **\***f**,** struct AST **\***ast**)** **{**

fprintf**(**f**,** "(%s"**,** UNOPS**[**ast**->**as\_unop**.**type**]);**

print**(**f**,** ast**->**as\_unop**.**operand**);**

fprintf**(**f**,** ")"**);**

**}**

void print\_lit**(**FILE **\***f**,** struct AST **\***ast**)** **{**

fprintf**(**f**,** "%" PRId64**,** ast**->**as\_literal**.**value**);**

**}**

static printer **\***ast\_printers**[]** **=** **{**

**[**AST\_BINOP**]** **=** print\_binop**,** **[**AST\_UNOP**]** **=** print\_unop**,** **[**AST\_LIT**]** **=** print\_lit**};**

void print**(**FILE **\***f**,** struct AST **\***ast**)** **{**

**if** **(**ast**)**

ast\_printers**[**ast**->**type**](**f**,** ast**);**

**else**

fprintf**(**f**,** "<NULL>"**);**

**}**

int main**()** **{**

struct AST **\***ast **=** **NULL;**

print**(**stdout**,** ast**);**

printf**(**"\n"**);**

struct AST**\*** \_999\_ **=** lit**(**999**);**

struct AST**\*** \_728\_ **=** lit**(**728**);**

struct AST**\*** plus **=** add**(**\_999\_**,** \_728\_**);**

print**(**stdout**,** plus**);**

printf**(**"\n"**);**

struct AST**\*** \_3\_ **=** lit**(**3**);**

struct AST**\*** \_5\_ **=** lit**(**5**);**

struct AST**\*** p3\_5 **=** add**(**\_3\_**,** \_5\_**);**

struct AST**\*** \_9\_ **=** lit**(**9**);**

struct AST**\*** \_7\_ **=** lit**(**7**);**

struct AST**\*** d9\_7 **=** div**(**\_9\_**,** \_7\_**);**

struct AST**\*** mult **=** mul**(**p3\_5**,** d9\_7**);**

print**(**stdout**,** mult**);**

**return** 0**;**

**}**

**gcc -o printer0 printer0.c**

#include "vector.h"

struct vector {

int64\_t\* start\_addr;

size\_t capacity;

size\_t length;

};

struct vector\* vector(size\_t initCapacity) {

struct vector\* vect = malloc(sizeof(struct vector));

vect->start\_addr = malloc(sizeof(int64\_t) \* initCapacity);

vect->capacity = initCapacity;

vect->length = 0;

return vect;

}

void add(struct vector\* vect, int64\_t value) {

if (vect->capacity == vect->length) {

vect->capacity = 2 \* vect->capacity;

vect->start\_addr = realloc(vect->start\_addr, vect->capacity \* sizeof(int64\_t));

}

vect->start\_addr[vect->length++] = value;

}

void add\_index(struct vector\* vect, size\_t index, int64\_t value) {

if (index > vect->capacity) { // maybe index + 1

vect->capacity = 2 \* index \* sizeof(int64\_t);

vect->start\_addr = realloc(vect->start\_addr, vect->capacity);

}

vect->length = index + 1;

vect->start\_addr[index] = value;

}

size\_t get\_capacity(struct vector\* vect) {

return vect->capacity;

}

void set\_capacity(struct vector\* vect, size\_t \_capacity) {

vect->start\_addr = realloc(vect->start\_addr, \_capacity \* sizeof(int64\_t));

vect->capacity = \_capacity;

}

size\_t get\_length(struct vector\* vect) {

return vect->length;

}

void print\_vector(FILE\* f, struct vector\* vect) {

fprintf(f, "\n [");

for (int i = 0; i < vect->length; i++) {

fprintf(f, "%" PRId64 " ", vect->start\_addr[i]);

}

fprintf(f, "]");

}

#include <inttypes.h>

#include "vector.h"

int main() {

struct vector\* vect = vector(5);

add(vect, 5);

add(vect, 1);

add(vect, 7);

print\_vector(stdout, vect);

add(vect, 9);

add(vect, 123);

add(vect, 34);

print\_vector(stdout, vect);

return 0;

}

#include <stdio.h>

#include <inttypes.h>

#include <malloc.h>

struct vector;

struct vector\* vector(size\_t);

// int64\_t get\_index(size\_t);

void add(struct vector\*, int64\_t);

void add\_index(struct vector\*, size\_t, int64\_t);

size\_t get\_capacity(struct vector\*);

void set\_capacity(struct vector\*, size\_t);

size\_t get\_length(struct vector\*);

void print\_vector(FILE\*, struct vector\*);

=====

#include <inttypes.h>

#include "array.h"

int main() {

struct array\_int\* arr = array\_int(5);

for (size\_t i = 0; i < 10; i++){

add(arr, (i\*i));

}

print\_arr(stdout, arr);

for (size\_t i = 0; i < 10; i++){

add(arr, 0);

}

print\_arr(stdout, arr);

return 0;

}

====

#include "array.h"

struct array\_int {

int64\_t\* start\_addr;

size\_t capacity;

size\_t length;

};

struct array\_int\* array\_int(size\_t startCapacity) {

struct array\_int\* arr = malloc(sizeof(struct array\_int));

arr->start\_addr = malloc(sizeof(int64\_t) \* startCapacity);

arr->capacity = startCapacity;

arr->length = 0;

return arr;

}

void add(struct array\_int\* arr, int64\_t value) {

if (arr->capacity == arr->length) {

arr->capacity = 2 \* arr->capacity;

arr->start\_addr = realloc(arr->start\_addr, arr->capacity \* sizeof(int64\_t));

}

arr->start\_addr[arr->length++] = value;

}

size\_t get\_capacity(struct array\_int\* arr) {

return arr->capacity;

}

void set\_capacity(struct array\_int\* arr, size\_t \_capacity) {

arr->start\_addr = realloc(arr->start\_addr, \_capacity \* sizeof(int64\_t));

arr->capacity = \_capacity;

}

size\_t get\_length(struct array\_int\* arr) {

return arr->length;

}

void print\_arr(FILE\* f, struct array\_int\* arr) {

fprintf(f, "\n [");

for (int i = 0; i < arr->length; i++) {

fprintf(f, "%" PRId64 " ", arr->start\_addr[i]);

}

fprintf(f, "]");

}

====

#include <stdio.h>

#include <inttypes.h>

#include <stdlib.h>

struct array\_int;

struct array\_int\* array\_int(size\_t);

void add(struct array\_int\*, int64\_t);

size\_t get\_capacity(struct array\_int\*);

void set\_capacity(struct array\_int\*, size\_t);

size\_t get\_length(struct array\_int\*);

void print\_arr(FILE\*, struct array\_int\*);

===

printer1.o: printer1.c ast.h

gcc -c -o printer1.o printer1.c

printer1: printer1.o

gcc -o printer1 printer1.o

bad.o: bad.c

gcc -c -o bad.o bad.c

bad: bad.o

gcc -o bad bad.o

array.o: array.c

gcc -c -o task1.o task1.c

task1.o: task1.c array.h

gcc -c -o task1.o task1.c

task1: task1.o

gcc -o task1 task1.o array.o

**Seminar 12**

The solution of the second exercise:

**Vector.h**

#include <stdio.h>

#include <inttypes.h>

#include <malloc.h>

struct vector;

struct vector\* vector(size\_t);

// int64\_t get\_index(size\_t);

void add(struct vector\*, int64\_t);

void add\_index(struct vector\*, size\_t, int64\_t);

size\_t get\_capacity(struct vector\*);

void set\_capacity(struct vector\*, size\_t);

size\_t get\_length(struct vector\*);

void print\_vector(FILE\*, struct vector\*);

**Vector.c**

#include "vector.h"

struct vector {

int64\_t\* start\_addr;

size\_t capacity;

size\_t length;

};

struct vector\* vector(size\_t initCapacity) {

struct vector\* vect = malloc(sizeof(struct vector));

vect->start\_addr = malloc(sizeof(int64\_t) \* initCapacity);

vect->capacity = initCapacity;

vect->length = 0;

return vect;

}

void add(struct vector\* vect, int64\_t value) {

if (vect->capacity == vect->length) {

vect->capacity = 2 \* vect->capacity;

vect->start\_addr = realloc(vect->start\_addr, vect->capacity \* sizeof(int64\_t));

}

vect->start\_addr[vect->length++] = value;

}

void add\_index(struct vector\* vect, size\_t index, int64\_t value) {

if (index > vect->capacity) { // maybe index + 1

vect->capacity = 2 \* index \* sizeof(int64\_t);

vect->start\_addr = realloc(vect->start\_addr, vect->capacity);

}

vect->length = index + 1;

vect->start\_addr[index] = value;

}

size\_t get\_capacity(struct vector\* vect) {

return vect->capacity;

}

void set\_capacity(struct vector\* vect, size\_t \_capacity) {

vect->start\_addr = realloc(vect->start\_addr, \_capacity \* sizeof(int64\_t));

vect->capacity = \_capacity;

}

size\_t get\_length(struct vector\* vect) {

return vect->length;

}

void print\_vector(FILE\* f, struct vector\* vect) {

fprintf(f, "\n [");

for (int i = 0; i < vect->length; i++) {

fprintf(f, "%" PRId64 " ", vect->start\_addr[i]);

}

fprintf(f, "]");

}

**Test.c**

#include <inttypes.h>

#include "vector.h"

int main() {

struct vector\* vect = vector(5);

add(vect, 5);

add(vect, 1);

add(vect, 7);

print\_vector(stdout, vect);

add(vect, 9);

add(vect, 123);

add(vect, 34);

print\_vector(stdout, vect);

return 0;

}

**Семинар 13**

**Question** What do the flags: -O0, -O1, -O2, -Os?  
 **Answer**: -O0 - optimization of the first level, second, third. The larger the number, the more optimizations the compiler makes. -Os - optimize size (optimize so that the program takes up as little memory as possible)

**Question** Show how, with an increase in the level of optimizations, reads from memory disappear:

**Answer:** We should look at QWORD PTR-type constructs that allocate memory. Having compiled the code with the -O0 and -O3 keys, we will clearly see a difference in the number of these constructs, which indicates a smaller amount of allocated memory.

**Question** mark the function print\_int as static. What happened in the optimized code and why?

**Answer:** the function is marked as static = not visible outside the module.

Without static:

Изображение выглядит как текст, монитор, снимок экрана, экран

Автоматически созданное описание

With static:

Изображение выглядит как текст, монитор, снимок экрана, экран

Автоматически созданное описание

The static keyword allows you to limit the scope of the function. How does she do it? It merges the code of the method with the code of the main program, which does not allow calling the method from outside our current file. I.e. methods with the static modifier are visible only for the file in which these methods are written. Actually, this is what we see in the example.

**Question** What does this instruction do? (leave)

**Answer:**

Изображение выглядит как текст

Автоматически созданное описание

**Question** Compile it without optimizations and explain the contents of the function maximum. Why rsp decreases by this number?

Because first the array elements are written to the red zone of the function, and then further... 128 (red zone) + 3992(array size) - 24 = 4096

**Вопрос:**

Compile the following code as optimized as possible.

Изображение выглядит как текст

Автоматически созданное описание

This function adds the second to the first argument twice; both arguments are pointers to numbers. We could add twice the second to the first argument, and that would be faster.

Look closely at the assembler function; is there this optimization? If so, why is it correct, if not, why is it incorrect?

**Answer:**

We may have a case where X and ADD will point to the same memory address, and then, in the case of adding the doubled value of ADD to X, we will get not X + 2\*ADD, but 4\*ADD, because we first double the value at the address, i.e. both X and ADD are doubled at once, and then we add the doubled original value at the same address. i.e., optimization does not occur.

**Question:**

Compile the following code as optimized as possible.

Изображение выглядит как текст

Автоматически созданное описание

**Answer:**

In the case of using the restrict keyword, we tell the compiler that only one variable can point to one memory area (the programmer guarantees this), and then we can use optimization.

**Question:**

\How will the compiled code with optimizations change? Read pages 281-282 in “Low-level programming” for the meaning of the keyword restrict and explain its effect on your code.

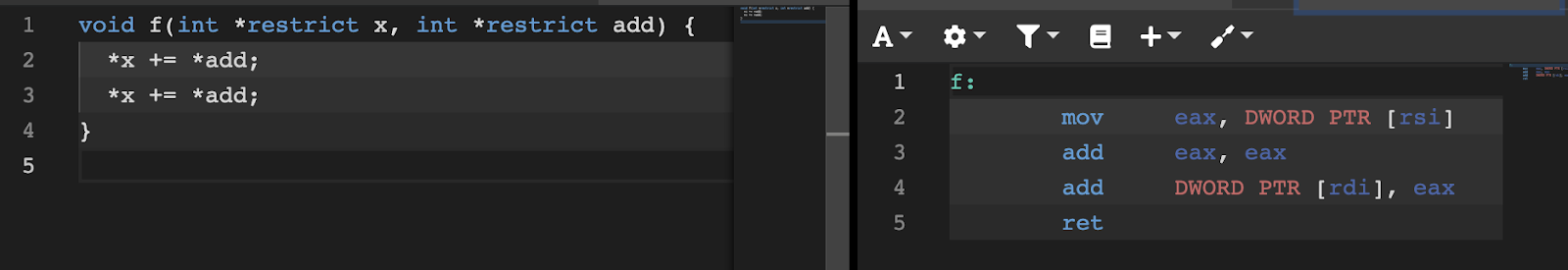
The restrict keyword allows the programmer to tell the compiler that the declared pointer addresses a memory area that is not referenced by any other pointer. The programmer gives a guarantee that more than one pointer will not be referenced to the memory section. At the same time, the optimizing compiler can generate more efficient code (see the example below).

**Without restrict:**

**Изображение выглядит как текст, монитор, снимок экрана

Автоматически созданное описание**

**With restrict:**

****