

warsztat4

Generated by Doxygen 1.9.3

1 File Index	1
1.1 File List	1
2 File Documentation	3
2.1 measurments.c File Reference	3
2.1.1 Function Documentation	3
2.1.1.1 main()	3
2.2 measurments.c	4
2.3 my_memset.c File Reference	5
2.3.1 Function Documentation	6
2.3.1.1 my_memset_ver1()	6
2.3.1.2 my_memset_ver2()	7
2.3.1.3 my_memset_ver3()	7
2.3.1.4 my_memset_ver4()	8
2.3.1.5 my_memset_ver5()	9
2.3.1.6 my_memset_ver6()	10
2.4 my_memset.c	11
2.5 my_memset.h	14
Index	15

Chapter 1

File Index

1.1 File List

Here is a list of all documented files with brief descriptions:

measurements.c	3
my_memset.c	5
my_memset.h	??

Chapter 2

File Documentation

2.1 measurments.c File Reference

Functions

- int [main](#) (void)

Główna funkcja, w której są wykonywane pomiary czasu działania poszczególnych implementacji funkcji memset.

2.1.1 Function Documentation

2.1.1.1 main()

```
int main (  
    void )
```

Główna funkcja, w której są wykonywane pomiary czasu działania poszczególnych implementacji funkcji memset.

Returns

int

Definition at line 29 of file [measurments.c](#).

```
00029     {  
00030  
00031         char buffer1[500], buffer2[1024*1024];  
00032  
00033         printf("##### \n");  
00034         measure_function(memset (&buffer1[0], '$', sizeof(buffer1)), "memset");  
00035         measure_function(memset (&buffer2[0], '$', sizeof(buffer2)), "memset");  
00036         printf("##### \n");  
00037  
00038         printf("\n");  
00039  
00040         memset (&buffer1[0], 0, sizeof(buffer1));  
00041         memset (&buffer2[0], 0, sizeof(buffer2));  
00042  
00043         printf("##### \n");  
00044         measure_function(my_memset_ver1 (&buffer1[0], '$', sizeof(buffer1)), "my_memset_ver1");  
00045         measure_function(my_memset_ver1 (&buffer2[0], '$', sizeof(buffer2)), "my_memset_ver1");  
00046         printf("##### \n");
```

```

00047
00048     printf("\n");
00049
00050     memset(&buffer1[0], 0, sizeof(buffer1));
00051     memset(&buffer2[0], 0, sizeof(buffer2));
00052
00053     printf("##### \n");
00054     measure_function(my_memset_ver2(&buffer1[0], '$', sizeof(buffer1)), "my_memset_ver2");
00055     measure_function(my_memset_ver2(&buffer2[0], '$', sizeof(buffer2)), "my_memset_ver2");
00056     printf("##### \n");
00057
00058     printf("\n");
00059
00060     memset(&buffer1[0], 0, sizeof(buffer1));
00061     memset(&buffer2[0], 0, sizeof(buffer2));
00062
00063     printf("##### \n");
00064     measure_function(my_memset_ver3(&buffer1[0], '$', sizeof(buffer1)), "my_memset_ver3");
00065     measure_function(my_memset_ver3(&buffer2[0], '$', sizeof(buffer2)), "my_memset_ver3");
00066     printf("##### \n");
00067
00068     printf("\n");
00069
00070     memset(&buffer1[0], 0, sizeof(buffer1));
00071     memset(&buffer2[0], 0, sizeof(buffer2));
00072
00073     printf("##### \n");
00074     measure_function(my_memset_ver4(&buffer1[0], '$', sizeof(buffer1)), "my_memset_ver4");
00075     measure_function(my_memset_ver4(&buffer2[0], '$', sizeof(buffer2)), "my_memset_ver4");
00076     printf("##### \n");
00077
00078     printf("\n");
00079
00080     memset(&buffer1[0], 0, sizeof(buffer1));
00081     memset(&buffer2[0], 0, sizeof(buffer2));
00082
00083     printf("##### \n");
00084     measure_function(my_memset_ver5(&buffer1[0], '$', sizeof(buffer1)), "my_memset_ver5");
00085     measure_function(my_memset_ver5(&buffer2[0], '$', sizeof(buffer2)), "my_memset_ver5");
00086     printf("##### \n");
00087
00088     printf("\n");
00089
00090     printf("##### \n");
00091     measure_function(my_memset_ver6(&buffer1[0], '$', sizeof(buffer1)), "my_memset_ver6");
00092     measure_function(my_memset_ver6(&buffer2[0], '$', sizeof(buffer2)), "my_memset_ver6");
00093     printf("##### \n");
00094
00095     printf("\n");
00096
00097     return 0;
00098 }

```

2.2 measurments.c

[Go to the documentation of this file.](#)

```

00001
00002 #include <stdio.h>
00003 #include <stdlib.h>
00004 #include <string.h>
00005 #include <time.h>
00006 #include "my_memset.h"
00007
00008 # define measure_function( function , label ) \
00009 do { \
00010 struct timespec start = {0}; \
00011 struct timespec end = {0}; \
00012 ( void ) clock_gettime ( CLOCK_MONOTONIC , &start); \
00013 ( void ) function ; \
00014 ( void ) clock_gettime ( CLOCK_MONOTONIC , &end); \
00015 const double seconds = end.tv_sec - start.tv_sec; \
00016 const double nanoseconds = end.tv_nsec - start.tv_nsec; \
00017 const double final_time = \
00018 (( seconds * 1e9 ) + nanoseconds ) * 1e-9; \
00019 printf ( " Measured time of function %s = %lf \n " , \
00020 label , final_time ) ; \
00021 } while (0)
00022
00029 int main(void){
00030
00031     char buffer1[500], buffer2[1024*1024];

```



```

00032
00033     printf("##### \n");
00034     measure_function(memset (&buffer1[0], '$', sizeof(buffer1)), "memset");
00035     measure_function(memset (&buffer2[0], '$', sizeof(buffer2)), "memset");
00036     printf("##### \n");
00037
00038     printf("\n");
00039
00040     memset (&buffer1[0], 0, sizeof(buffer1));
00041     memset (&buffer2[0], 0, sizeof(buffer2));
00042
00043     printf("##### \n");
00044     measure_function(my_memset_ver1 (&buffer1[0], '$', sizeof(buffer1)), "my_memset_ver1");
00045     measure_function(my_memset_ver1 (&buffer2[0], '$', sizeof(buffer2)), "my_memset_ver1");
00046     printf("##### \n");
00047
00048     printf("\n");
00049
00050     memset (&buffer1[0], 0, sizeof(buffer1));
00051     memset (&buffer2[0], 0, sizeof(buffer2));
00052
00053     printf("##### \n");
00054     measure_function(my_memset_ver2 (&buffer1[0], '$', sizeof(buffer1)), "my_memset_ver2");
00055     measure_function(my_memset_ver2 (&buffer2[0], '$', sizeof(buffer2)), "my_memset_ver2");
00056     printf("##### \n");
00057
00058     printf("\n");
00059
00060     memset (&buffer1[0], 0, sizeof(buffer1));
00061     memset (&buffer2[0], 0, sizeof(buffer2));
00062
00063     printf("##### \n");
00064     measure_function(my_memset_ver3 (&buffer1[0], '$', sizeof(buffer1)), "my_memset_ver3");
00065     measure_function(my_memset_ver3 (&buffer2[0], '$', sizeof(buffer2)), "my_memset_ver3");
00066     printf("##### \n");
00067
00068     printf("\n");
00069
00070     memset (&buffer1[0], 0, sizeof(buffer1));
00071     memset (&buffer2[0], 0, sizeof(buffer2));
00072
00073     printf("##### \n");
00074     measure_function(my_memset_ver4 (&buffer1[0], '$', sizeof(buffer1)), "my_memset_ver4");
00075     measure_function(my_memset_ver4 (&buffer2[0], '$', sizeof(buffer2)), "my_memset_ver4");
00076     printf("##### \n");
00077
00078     printf("\n");
00079
00080     memset (&buffer1[0], 0, sizeof(buffer1));
00081     memset (&buffer2[0], 0, sizeof(buffer2));
00082
00083     printf("##### \n");
00084     measure_function(my_memset_ver5 (&buffer1[0], '$', sizeof(buffer1)), "my_memset_ver5");
00085     measure_function(my_memset_ver5 (&buffer2[0], '$', sizeof(buffer2)), "my_memset_ver5");
00086     printf("##### \n");
00087
00088     printf("\n");
00089
00090     printf("##### \n");
00091     measure_function(my_memset_ver6 (&buffer1[0], '$', sizeof(buffer1)), "my_memset_ver6");
00092     measure_function(my_memset_ver6 (&buffer2[0], '$', sizeof(buffer2)), "my_memset_ver6");
00093     printf("##### \n");
00094
00095     printf("\n");
00096
00097     return 0;
00098 }

```

2.3 my_memset.c File Reference

Functions

- void * [my_memset_ver1](#) (void *ptr, int value, size_t num)

Funkcja my_memset_ver1 - pierwsza wersja, czyli ustawianie każdego bajtu na zadaną wartość po kolei (bajt po bajcie).

- void * [my_memset_ver2](#) (void *ptr, int value, size_t num)

Funkcja memset (wersja 2), w której wykorzystano instrukcje wektorowe (kopiuje 32 bajty do tablicy naraz).

- void * [my_memset_ver3](#) (void *ptr, int value, size_t num)

Funkcja memset (wersja 3) z rozwiniętą pętlą (możliwe zwiększenie efektywności kodu).

- void * [my_memset_ver4](#) (void *restrict ptr, int value, size_t num)

Funkcja memset (wersja 4), w której wykorzystano instrukcje wektorowe (kopiuje 32 bajty to tablicy naraz) oraz słowo kluczowe restrict (możliwość optymalizacji przez kompilator).

- void * [my_memset_ver5](#) (void *ptr, int value, size_t num)

Funkcja memset (wersja 5) - kopiuje po 64 bajty do tablicy w jednym przebiegu pętli (za pomocą instrukcji wektorowych).

- void * [my_memset_ver6](#) (void *ptr, int value, size_t num)

Funkcja memset (wersja 6) - kopiuje po 128 bajtów do tablicy w jednym przebiegu pętli (za pomocą instrukcji wektorowych).

2.3.1 Function Documentation

2.3.1.1 my_memset_ver1()

```
void * my_memset_ver1 (
    void * ptr,
    int value,
    size_t num )
```

Funkcja my_memset_ver1 - pierwsza wersja, czyli ustawianie każdego bajtu na zadaną wartość po kolei (bajt po bajcie).

Parameters

in	<i>ptr</i>	
in	<i>value</i>	
in	<i>num</i>	

Returns

void*

Definition at line 15 of file [my_memset.c](#).

```
00015                                     {
00016
00017     if(ptr == NULL){
00018         return NULL;
00019     }
00020
00021     unsigned char* cptr = (unsigned char*)ptr;
00022
00023     for(size_t i = 0 ; i < num ; ++i){
00024         (*cptr) = (unsigned char)value;
00025         cptr += 1;
00026     }
00027
00028     return ptr;
00029 }
00030
00031
00032
00033 }
```

2.3.1.2 my_memset_ver2()

```
void * my_memset_ver2 (
    void * ptr,
    int value,
    size_t num )
```

Funkcja memset (wersja 2), w której wykorzystano instrukcje wektorowe (kopiuje 32 bajty do tablicy naraz).

Parameters

in	<i>ptr</i>	
in	<i>value</i>	
in	<i>num</i>	

Returns

void*

Definition at line 43 of file [my_memset.c](#).

```
00043                                     {
00044
00045     if(ptr == NULL){
00046         return NULL;
00047     }
00048
00049     __m256i v = _mm256_set1_epi8((unsigned char)value);
00050
00051     size_t number_of_iterations = num/sizeof(__m256i);
00052     size_t rest = num%sizeof(__m256i);
00053
00054     for(size_t i = 0 ; i < number_of_iterations ; ++i){
00055
00056         _mm256_storeu_si256((__m256i*)ptr, v);
00057
00058         ptr += sizeof(__m256i);
00059     }
00060
00061     unsigned char* cptr2 = (unsigned char*)ptr;
00062
00063     for(size_t i = 0 ; i < rest ; ++i){
00064
00065         (*cptr2) = (unsigned char)value;
00066
00067         cptr2 += 1;
00068     }
00069
00070     return ptr;
00071
00072
00073
00074 }
```

2.3.1.3 my_memset_ver3()

```
void * my_memset_ver3 (
    void * ptr,
    int value,
    size_t num )
```

Funkcja memset (wersja 3) z rozwiniętą pętlą (możliwe zwiększenie efektywności kodu).

Parameters

in	<i>ptr</i>	
in	<i>value</i>	
in	<i>num</i>	

Returns

void*

Definition at line 84 of file [my_memset.c](#).

```

00084                                     {
00085
00086     if(ptr == NULL){
00087         return NULL;
00088     }
00089
00090     unsigned char* _ptr = (unsigned char*)ptr;
00091
00092     unsigned char casted_value = (unsigned char)value;
00093
00094     size_t batch_size = 16;
00095
00096     size_t number_of_iterations = num/batch_size;
00097     size_t rest = num%batch_size;
00098
00099     size_t p = 0;
00100
00101     for(size_t i = 0 ; i < number_of_iterations ; i++){
00102
00103         _ptr[p] = casted_value;
00104         _ptr[p + 1] = casted_value;
00105         _ptr[p + 2] = casted_value;
00106         _ptr[p + 3] = casted_value;
00107         _ptr[p + 4] = casted_value;
00108         _ptr[p + 5] = casted_value;
00109         _ptr[p + 6] = casted_value;
00110         _ptr[p + 7] = casted_value;
00111         _ptr[p + 8] = casted_value;
00112         _ptr[p + 9] = casted_value;
00113         _ptr[p + 10] = casted_value;
00114         _ptr[p + 11] = casted_value;
00115         _ptr[p + 12] = casted_value;
00116         _ptr[p + 13] = casted_value;
00117         _ptr[p + 14] = casted_value;
00118         _ptr[p + 15] = casted_value;
00119
00120         p += 16;
00121     }
00122
00123     _ptr += number_of_iterations*16;
00124
00125     for(size_t i = 0 ; i < rest ; ++i){
00126
00127         _ptr[i] = casted_value;
00128     }
00129
00130 }
00131
00132 return ptr;
00133 }

```

2.3.1.4 my_memset_ver4()

```

void * my_memset_ver4 (
    void *restrict ptr,
    int value,
    size_t num )

```

Funkcja memset (wersja 4), w której wykorzystano instrukcje wektorowe (kopiuje 32 bajty to tablicy naraz) oraz słowo kluczowe restrict (możliwość optymalizacji przez kompilator).

Parameters

in	<i>ptr</i>	
in	<i>value</i>	
in	<i>num</i>	

Returns

void*

Definition at line 144 of file my_memset.c.

```

00144                                     {
00145
00146     if(ptr == NULL){
00147         return NULL;
00148     }
00149
00150     __m256i v = _mm256_set1_epi8((unsigned char)value);
00151
00152     size_t number_of_iterations = num/sizeof(__m256i);
00153     size_t rest = num%sizeof(__m256i);
00154
00155     for(size_t i = 0 ; i < number_of_iterations ; ++i){
00156         _mm256_storeu_si256((__m256i*)ptr, v);
00157
00158         ptr += sizeof(__m256i);
00159     }
00160
00161     unsigned char* cptr2 = (unsigned char*)ptr;
00162
00163     for(size_t i = 0 ; i < rest ; ++i){
00164         (*cptr2) = (unsigned char)value;
00165         cptr2 += 1;
00166     }
00167
00168     return ptr;
00169 }
00170
00171
00172
00173
00174
00175 }
```

2.3.1.5 my_memset_ver5()

```

void * my_memset_ver5 (
    void * ptr,
    int value,
    size_t num )
```

Funkcja memset (wersja 5) - kopiuje po 64 bajty do tablicy w jednym przebiegu pętli (za pomocą instrukcji wektorowych).

Parameters

in	<i>ptr</i>	
in	<i>value</i>	
in	<i>num</i>	

Returns

void*

Definition at line 186 of file [my_memset.c](#).

```

00186                                     {
00187
00188     if(ptr == NULL){
00189         return NULL;
00190     }
00191
00192     __m256i v1 = _mm256_set1_epi8((unsigned char) value);
00193
00194     size_t number_of_iterations = (num)/(2*sizeof(__m256i));
00195     size_t rest = (num)%(2*sizeof(__m256i));
00196
00197     for(size_t i = 0 ; i < number_of_iterations ; i++){
00198
00199         _mm256_storeu_si256((__m256i*)ptr, v1);
00200
00201         ptr += sizeof(__m256i);
00202
00203         _mm256_storeu_si256((__m256i*)ptr, v1);
00204
00205         ptr += sizeof(__m256i);
00206
00207     }
00208
00209     unsigned char* cptr = (unsigned char*)ptr;
00210
00211     for(size_t i = 0 ; i < rest ; i++){
00212         *cptr = (unsigned char)value;
00213
00214         cptr += 1;
00215     }
00216
00217     return ptr;
00218
00219 }

```

2.3.1.6 my_memset_ver6()

```

void * my_memset_ver6 (
    void * ptr,
    int value,
    size_t num )

```

Funkcja memset (wersja 6) - kopiuje po 128 bajtów do tablicy w jednym przebiegu pętli (za pomocą instrukcji wektorowych).

Parameters

in	<i>ptr</i>	
in	<i>value</i>	
in	<i>num</i>	

Returns

void*

Definition at line 230 of file [my_memset.c](#).

```

00230                                     {
00231

```

```

00232     if(ptr == NULL){
00233         return NULL;
00234     }
00235
00236     __m256i v1 = _mm256_set1_epi8((unsigned char) value);
00237
00238     size_t number_of_iterations = (num)/(4*sizeof(__m256i));
00239     size_t rest = (num)%(4*sizeof(__m256i));
00240
00241     for(size_t i = 0 ; i < number_of_iterations ; i++){
00242
00243         _mm256_storeu_si256((__m256i*)ptr, v1);
00244
00245         ptr += sizeof(__m256i);
00246
00247         _mm256_storeu_si256((__m256i*)ptr, v1);
00248
00249         ptr += sizeof(__m256i);
00250
00251         _mm256_storeu_si256((__m256i*)ptr, v1);
00252
00253         ptr += sizeof(__m256i);
00254
00255         _mm256_storeu_si256((__m256i*)ptr, v1);
00256
00257         ptr += sizeof(__m256i);
00258
00259     }
00260
00261     unsigned char* cptr = (unsigned char*)ptr;
00262
00263     for(size_t i = 0 ; i < rest ; i++){
00264         *cptr = (unsigned char)value;
00265
00266         cptr += 1;
00267     }
00268
00269     return ptr;
00270
00271 }

```

2.4 my_memset.c

[Go to the documentation of this file.](#)

```

00001
00002 #include <stdio.h>
00003 #include <immintrin.h>
00004 #include <stdint.h>
00005 #include <string.h>
00006
00015 void* my_memset_ver1(void* ptr, int value, size_t num){
00016
00017     if(ptr == NULL){
00018         return NULL;
00019     }
00020
00021     unsigned char* cptr = (unsigned char*)ptr;
00022
00023     for(size_t i = 0 ; i < num ; ++i){
00024
00025         (*cptr) = (unsigned char)value;
00026
00027         cptr += 1;
00028
00029     }
00030
00031     return ptr;
00032
00033 }
00034
00043 void* my_memset_ver2(void* ptr, int value, size_t num){
00044
00045     if(ptr == NULL){
00046         return NULL;
00047     }
00048
00049     __m256i v = _mm256_set1_epi8((unsigned char)value);
00050
00051     size_t number_of_iterations = num/sizeof(__m256i);
00052     size_t rest = num%sizeof(__m256i);
00053

```

```

00054     for(size_t i = 0 ; i < number_of_iterations ; ++i){
00055         __mm256_storeu_si256((__m256i*)ptr, v);
00056         ptr += sizeof(__m256i);
00057     }
00058     unsigned char* cptr2 = (unsigned char*)ptr;
00059     for(size_t i = 0 ; i < rest ; ++i){
00060         (*cptr2) = (unsigned char)value;
00061         cptr2 += 1;
00062     }
00063     return ptr;
00064 }
00065
00066 void* my_memset_ver3(void* ptr, int value, size_t num){
00067     if(ptr == NULL){
00068         return NULL;
00069     }
00070     unsigned char* _ptr = (unsigned char*)ptr;
00071     unsigned char casted_value = (unsigned char)value;
00072     size_t batch_size = 16;
00073     size_t number_of_iterations = num/batch_size;
00074     size_t rest = num%batch_size;
00075     size_t p = 0;
00076     for(size_t i = 0 ; i < number_of_iterations ; i++){
00077         _ptr[p] = casted_value;
00078         _ptr[p + 1] = casted_value;
00079         _ptr[p + 2] = casted_value;
00080         _ptr[p + 3] = casted_value;
00081         _ptr[p + 4] = casted_value;
00082         _ptr[p + 5] = casted_value;
00083         _ptr[p + 6] = casted_value;
00084         _ptr[p + 7] = casted_value;
00085         _ptr[p + 8] = casted_value;
00086         _ptr[p + 9] = casted_value;
00087         _ptr[p + 10] = casted_value;
00088         _ptr[p + 11] = casted_value;
00089         _ptr[p + 12] = casted_value;
00090         _ptr[p + 13] = casted_value;
00091         _ptr[p + 14] = casted_value;
00092         _ptr[p + 15] = casted_value;
00093         p += 16;
00094     }
00095     _ptr += number_of_iterations*16;
00096     for(size_t i = 0 ; i < rest ; ++i){
00097         _ptr[i] = casted_value;
00098     }
00099     return ptr;
00100 }
00101
00102 void* my_memset_ver4(void* restrict ptr, int value, size_t num){
00103     if(ptr == NULL){
00104         return NULL;
00105     }
00106     __m256i v = __mm256_set1_epi8((unsigned char)value);
00107     size_t number_of_iterations = num/sizeof(__m256i);
00108     size_t rest = num%sizeof(__m256i);
00109     for(size_t i = 0 ; i < number_of_iterations ; ++i){
00110         __mm256_storeu_si256((__m256i*)ptr, v);

```



```
00158
00159     ptr += sizeof(__m256i);
00160
00161 }
00162
00163 unsigned char* cptr2 = (unsigned char*)ptr;
00164
00165 for(size_t i = 0 ; i < rest ; ++i){
00166
00167     (*cptr2) = (unsigned char)value;
00168
00169     cptr2 += 1;
00170
00171 }
00172
00173 return ptr;
00174
00175 }
00176
00186 void* my_memset_ver5(void* ptr, int value, size_t num){
00187
00188     if(ptr == NULL){
00189         return NULL;
00190     }
00191
00192     __m256i v1 = _mm256_set1_epi8((unsigned char) value);
00193
00194     size_t number_of_iterations = (num)/(2*sizeof(__m256i));
00195     size_t rest = (num)%(2*sizeof(__m256i));
00196
00197     for(size_t i = 0 ; i < number_of_iterations ; i++){
00198
00199         _mm256_storeu_si256((__m256i*)ptr, v1);
00200
00201         ptr += sizeof(__m256i);
00202
00203         _mm256_storeu_si256((__m256i*)ptr, v1);
00204
00205         ptr += sizeof(__m256i);
00206
00207     }
00208
00209     unsigned char* cptr = (unsigned char*)ptr;
00210
00211     for(size_t i = 0 ; i < rest ; i++){
00212         *cptr = (unsigned char)value;
00213
00214         cptr += 1;
00215     }
00216
00217     return ptr;
00218
00219 }
00220
00230 void* my_memset_ver6(void* ptr, int value, size_t num){
00231
00232     if(ptr == NULL){
00233         return NULL;
00234     }
00235
00236     __m256i v1 = _mm256_set1_epi8((unsigned char) value);
00237
00238     size_t number_of_iterations = (num)/(4*sizeof(__m256i));
00239     size_t rest = (num)%(4*sizeof(__m256i));
00240
00241     for(size_t i = 0 ; i < number_of_iterations ; i++){
00242
00243         _mm256_storeu_si256((__m256i*)ptr, v1);
00244
00245         ptr += sizeof(__m256i);
00246
00247         _mm256_storeu_si256((__m256i*)ptr, v1);
00248
00249         ptr += sizeof(__m256i);
00250
00251         _mm256_storeu_si256((__m256i*)ptr, v1);
00252
00253         ptr += sizeof(__m256i);
00254
00255         _mm256_storeu_si256((__m256i*)ptr, v1);
00256
00257         ptr += sizeof(__m256i);
00258
00259     }
00260
00261     unsigned char* cptr = (unsigned char*)ptr;
00262
```

```
00263     for(size_t i = 0 ; i < rest ; i++){
00264         *cptr = (unsigned char)value;
00265
00266         cptr += 1;
00267     }
00268
00269     return ptr;
00270
00271 }
00272
```

2.5 my_memset.h

```
00001 #ifndef MY_MEMSET_H
00002 #define MY_MEMSET_H
00003
00004 void* my_memset_ver1(void* ptr, int value, size_t num);
00005 void* my_memset_ver2(void* ptr, int value, size_t num);
00006 void* my_memset_ver3(void* ptr, int value, size_t num);
00007 void* my_memset_ver4(void* ptr, int value, size_t num);
00008 void* my_memset_ver5(void* ptr, int value, size_t num);
00009 void* my_memset_ver6(void* ptr, int value, size_t num);
00010
00011 #endif
```

Index

- main
 - measurments.c, [3](#)
- measurments.c, [3](#)
 - main, [3](#)
- my_memset.c, [5](#)
 - my_memset_ver1, [6](#)
 - my_memset_ver2, [6](#)
 - my_memset_ver3, [7](#)
 - my_memset_ver4, [8](#)
 - my_memset_ver5, [9](#)
 - my_memset_ver6, [10](#)
- my_memset_ver1
 - my_memset.c, [6](#)
- my_memset_ver2
 - my_memset.c, [6](#)
- my_memset_ver3
 - my_memset.c, [7](#)
- my_memset_ver4
 - my_memset.c, [8](#)
- my_memset_ver5
 - my_memset.c, [9](#)
- my_memset_ver6
 - my_memset.c, [10](#)