

# Ukazatele, paměťové třídy, volání funkcí

Jan Faigl

Katedra počítačů  
Fakulta elektrotechnická  
České vysoké učení technické v Praze

Přednáška 05

B0B36PRP – Procedurální programování

Jan Faigl, 2023

B0B36PRP – Přednáška 05: Paměťové třídy

1 / 48

Modifikátor const a ukazatele

Část I

## Část 1 – Ukazatele a dynamická alokace

### Přehled témat

- Část 1 – Ukazatele a dynamická alokace
  - Modifikátor `const` a ukazatele
  - Dynamická alokace paměti

S. G. Kochan: kapitoly 8 a 11

- Část 2 – Paměťové třídy a volání funkcí
  - Výpočetní prostředky a běh programu
  - Rozsah platnosti proměnných
  - Paměťové třídy

S. G. Kochan: kapitola 8 a 11

- Část 3 – Zadání 5. domácího úkolu (HW05)

Jan Faigl, 2023

B0B36PRP – Přednáška 05: Paměťové třídy

2 / 48

Dynamická alokace paměti

Modifikátor const a ukazatele

Dynamická alokace paměti

### Modifikátor typu const

- Uvedením klíčového slova `const` můžeme označit proměnnou jako konstantu.  
*Překladač nás kontroluje, zdali se snažíme hodnotu proměnné změnit.*
- Definovat konstantu můžeme např.  
`const float pi = 3.14159265f;`
- Symbolická konstanta  
`#define PI 3.14159265`
- je pojmenování literálu, ve zdrojovém souboru je výkyt `PI` textově nahrazen literálem.

Připomínka

Jan Faigl, 2023

B0B36PRP – Přednáška 05: Paměťové třídy

3 / 48

Jan Faigl, 2023

B0B36PRP – Přednáška 05: Paměťové třídy

5 / 48

## Ukazatele na konstantní proměnné a konstantní ukazatele

- Klíčové slovo `const` můžeme zapsat před jméno proměnné nebo před `*` (typ/).
- Dostáváme 3 možnosti jak definovat ukazatel s `const`.
  - (a) `const int *ptr;` – ukazatel na konstantní proměnnou.
    - Nemůžeme použít pointer pro změnu hodnoty proměnné.
  - (b) `int *const ptr;` – konstantní ukazatel (`const` před jménem proměnné a mezi `*`).
    - Pointer nemůžeme nastavit na jinou adresu než tu při inicializaci.
  - (c) `const int *const ptr;` – konstantní ukazatel na konstantní hodnotu.
    - Kombinuje předchozí dva případy.
- Další alternativy zápisu (a) a (c) jsou
  - `const int * lze též zapsat jako int const *; const je stále před *.`
  - `const int * const lze též zapsat jako int const * const.`
  - `const` může být vlevo nebo vpravo od jména typu.
- Nebo komplexnější definice, např. `int ** const ptr;` – konstantní ukazatel na ukazatel na `int`.

lec05/const\_pointers.c

## Příklad – Konstantní ukazatel

- Hodnotu konstantního ukazatele nelze po inicializaci měnit.
- Zápis `int *const ptr;` můžeme číst zprava doleva:
  - `ptr` – proměnná, která je;
  - `*const` – konstantním ukazatelem;
  - `int` – na proměnnou typu `int`.

```

1 int v = 10;
2 int v2 = 20;
3 int *const ptr = &v;
4 printf("v: %d *ptr: %d\n", v, *ptr);
6 *ptr = 11; /* We can modify addressed value */
7 printf("v: %d\n", v);
9 ptr = &v2; /* IT IS NOT ALLOWED! */

```

lec05/const\_pointers.c

## Příklad – Ukazatel na konstantní proměnnou (hodnotu)

- Prostřednictvím ukazatele na konstantní proměnnou nemůžeme tuto proměnnou měnit.

```

1 int v = 10;
2 int v2 = 20;
4 const int *ptr = &v; // ptr cannot be used to modify v
5 printf("*ptr: %d\n", *ptr);
7 *ptr = 11; /* IT IS NOT ALLOWED! */
9 v = 11; /* We can modify the original variable */
10 printf("*ptr: %d\n", *ptr);
12 ptr = &v2; /* We can assign new address to ptr */
13 printf("*ptr: %d\n", *ptr);

```

lec05/const\_pointers.c

## Příklad – Konstantní ukazatel na konstantní proměnnou

- Hodnotu konstantního ukazatele na konstantního proměnnou nelze po inicializaci měnit a ani nelze prostřednictvím takového ukazatele měnit hodnotu adresované proměnné.
- Zápis `const int *const ptr;` čteme "zprava doleva":
  - `ptr` – proměnná, která je;
  - `*const` – konstantním ukazatelem;
  - `const int` – na proměnnou typu `const int`.

```

1 int v = 10;
2 int v2 = 20;
3 const int *const ptr = &v;
5 printf("v: %d *ptr: %d\n", v, *ptr);
7 ptr = &v2; /* IT IS NOT ALLOWED! */
8 *ptr = 11; /* IT IS NOT ALLOWED! */

```

lec05/const\_pointers.c

## Konstantní ukazatel (na konstantní hodnotu)

Příklad	Konstantní hodnota	Konstantní ukazatel	Popis
char *ptr	Ne	Ne	„ptr je ukazatel (*) na hodnotu char.“
const char *ptr	Ano	Ne	„ptr je ukazatel na hodnotu char konstantní.“
char const *ptr	Ano	Ne	„ptr je ukazatel na konstantní hodnotu char.“
char* const ptr	Ne	Ano	„ptr je konstantní ukazatel na hodnotu char.“
const char *const ptr	Ano	Ano	„ptr je konstantní ukazatel na hodnotu char konstantní.“

- Konstantní ukazatel je proměnná, jejíž hodnotu nemohu měnit. Ukazatel odkazuje na (stejné) paměťové místo, které mohu případně měnit.
- Konstantní hodnotu nemohu měnit. Tedy nemohu měnit obsah paměťového místa, na které odkazuje ukazatel (jehož adresa je uloženo v proměnné typu ukazatel).

## Příklad – Ukazatel na funkci 1/2

- Používáme dereferenční operátor \* podobně jako u proměnných.

```
double do_nothing(int v); /* function prototype */
double (*function_p)(int v); /* pointer to function */
function_p = do_nothing; /* assign the pointer */
(*function_p)(10); /* call the function */
```

- Závorky (\*function\_p) „pomáhají“ čist definici ukazatele.

Můžeme si představit, že závorky reprezentují jméno funkce. Definice proměnné ukazatel na funkci se tak v zásadě neliší od prototypu funkce.

- Podobně je volání funkce přes ukazatel na funkci identické běžnému volání funkce, kde místo jména funkce vystupuje jméno ukazatele na funkci.

## Ukazatel na funkci

- Implementace funkce je umístěna někde v paměti a podobně jako na proměnnou v paměti může ukazatel odkazovat na paměťové místo s definicí funkce.
- Můžeme definovat **ukazatel na funkci** a dynamicky volat funkci dle aktuální hodnoty ukazatele.
- Součástí volání funkce jsou předávané argumenty, které jsou též součástí typu ukazatele na funkci, resp. typy argumentů.
- Funkce (a volání funkce) je identifikátor funkce a () , tj.  
`typ_návratové_hodnoty funkce(argumenty funkce);`
- Ukazatel na funkci definujeme jako  
`typ_návratové_hodnoty (*ukazatel)(argumenty funkce);`

## Příklad – Ukazatel na funkci 2/2

- V případě funkce vracející ukazatel postupujeme identicky.

```
double* compute(int v);
double* (*function_p)(int v);
~~~~~----- substitute a function name
function_p = compute;
```

- Příklad použití ukazatele na funkci – [lec05/pointer\\_fnc.c](#)

- Ukazatele na funkce umožňují realizovat dynamickou vazbu volání funkce identifikované za běhu programu.

Ukazatel na funkci se může hodit v implementaci HW05 povinná a volitelné zadání. Při vhodném návrhu programu je základní část společná, „jen“ zaměníme funkci pro porovnávání dvou řetězců s využitím Hammingovy nebo Levenshtejnovy vzdálenosti. V případě obou funkcí může být vstup dva textové řetězce, případně včetně délky. Tedy můžeme jednoduše zaměnit ukazatel na funkci.

## Příklad použití ukazatale na funkci

- Vhodným využitím ukazatele na funkci je zajištění přístupu k datům pro jinak naprostě identický algoritmus, jako je řazení (funkce `qsort` z `stdlib.h`). Zejména pro pole hodnot složeného typu.

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

```
1 #include <stdio.h>
2 #include <stdlib.h>
4 void print(int n, int array[n]);
5 int compare(const void *pa, const void *pb);
7 int main(void)
8 {
9     const int n = 10;
10    int array[n];
11    for (int i = 0; i < n; ++i) {
12        array[i] = rand() % 100;
13    }
14    print(n, array);
15    qsort(array, n, sizeof(array[0]), compare);
16    print(n, array);
17    return 0;
18 }
```

```
20 void print(int n, int array[n])
21 {
22     for(int i = 0; i < n; ++i) {
23         i > 0 ? printf(", ") : 0;
24         printf("%d", array[i]);
25     }
26     n > 0 ? putchar('\n') : 0;
27 }
29 int compare(const void *pa, const void *pb)
30 {
31     const int a = *(int*)pa;
32     const int b = *(int*)pb;
33     return (a < b) - (a > b);
34 }
```

lec05/demo-pointer\_fnc.c

## Dynamická alokace paměti

- Přidělení bloku paměti velikosti `size` lze realizovat funkcí
 

```
void* malloc(size);
```

 Z knihovny `<stdlib.h>`
  - Velikost alokované paměti je uložena ve správci paměti.
  - Velikost není součástí ukazatele.**
  - Návratová hodnota je typu `void*` – přetypování nutné/vhodné.
  - Je plně na uživateli (programátorovi), jak bude s pamětí zacházet.

■ Příklad alokace paměti pro 10 proměnných typu `int`.

```
1 int *int_array;
2 int_array = (int*)malloc(10 * sizeof(int));
```

■ Operace s více hodnotami v paměťovém bloku je podobná poli.

■ Používáme pointerovou aritmetiku.

■ **Uvolnění paměti**

```
void free(pointer);
```

- Správce paměti uvolní paměť asociovanou k ukazateli.
- Hodnotu ukazatele však nemění!

*Stále obsahuje předešlou adresu, která však již není platná.*

## Definice typu – `typedef`

- Operátor `typedef` umožňuje definovat nový datový typ.
- Slouží k pojmenování typů, např. ukazatele, struktury a uniony.
- Struktury a uniony viz přednáška 6.*
- Například typ pro ukazatele na `double` a nové jméno pro `int`:

```
1 typedef double* double_p;
2 typedef int integer;
3 double_p x, y;
4 integer i, j;
```

- je totožné s použitím původních typů

```
1 double *x, *y;
2 int i, j;
```

- Zavedením typů operátorem `typedef`, např. v hlavičkovém souboru, umožňuje systematické používání nových jmen typů v celém programu.
- Viz např. `<inttypes.h>`.*
- Výhoda zavedení nových typů je především u složitějších typů jako jsou ukazatele na funkce nebo struktury.

## Příklad alokace dynamické paměti 1/3

- Alokace se nemusí nutně povést – testujeme návratovou hodnotu funkce `malloc()`.
- Pro vyplnění adresy alokované paměti předáváme proměnnou jako ukazatel na proměnnou typu ukazatel na `int`.

```
1 void* allocate_memory(int size, void **ptr)
2 {
3     // use **ptr to store value of newlly allocated
4     // memory in the pointer ptr (i.e., the address the
5     // pointer ptr is pointed).
6     // call library function malloc to allocate memory
7     *ptr = malloc(size);
8     if (*ptr == NULL) {
9         fprintf(stderr, "Error: allocation fail");
10        exit(-1); /* exit program if allocation fail */
11    }
12    return *ptr;
13 }
```

lec05/malloc\_demo.c

## Příklad alokace dynamické paměti 2/3

- Pro vyplnění hodnot pole alokovaného dynamicky nám postačuje předávat hodnotu adresy paměti pole.

```
1 void fill_array(int size, int* array)
2 {
3     for (int i = 0; i < size; ++i) {
4         *(array++) = random();
5     }
6 }
```

- Po uvolnění paměti odkazuje ukazatel stále na původní adresu, proto můžeme explicitně nulovat.

*Předání ukazatele na ukazatele je nutné, jinak nemůžeme nulovat.*

```
1 void deallocate_memory(void **ptr)
2 {
3     if (ptr != NULL && *ptr != NULL) {
4         free(*ptr);
5         *ptr = NULL;
6     }
7 }
```

lec05/malloc\_demo.c

## Příklad - Načítání textového řetězce 1/3

- Implementujete načtení libovolně dlouhého řádku ze `stdin`.
- Řádek je zakončen znakem nového řádku `'\n'`, který **není součástí načteného vstupu**.
- Reportujte chybové stavy `ERROR_IN = 100` a `ERROR_MEM = 101`.
- Po úspěšném načtení vstupu, reportujte velikost vstupu voláním funkce `strlen()` z `string.h`.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #ifndef INIT_SIZE
5 #define INIT_SIZE 128
6#endif
7 enum {
8     ERROR_OK = EXIT_SUCCESS,
9     ERROR_IN = 100,
10    ERROR_MEM = 101,
11 };
12
13 char* read(int *error);
14
15 char* handle_str(char r, size_t l, char *str, int *error)
16 {
17     if (*str == '\0') {
18         *str = r;
19         if (r == '\n') {
20             *error = ERROR_IN;
21             return str;
22         }
23         else {
24             *str = '\0';
25             *error = ERROR_MEM;
26             return str;
27         }
28     }
29     else {
30         *str = r;
31         if (r == '\n') {
32             *str = '\0';
33             *error = ERROR_IN;
34             return str;
35         }
36         else {
37             *str = r;
38             if (*str == '\0') {
39                 *error = ERROR_MEM;
40                 return str;
41             }
42             else {
43                 *str = r;
44                 if (r == '\n') {
45                     *str = '\0';
46                     *error = ERROR_IN;
47                     return str;
48                 }
49                 else {
50                     *str = r;
51                     if (*str == '\0') {
52                         *error = ERROR_MEM;
53                         return str;
54                     }
55                 }
56             }
57         }
58     }
59 }
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
627
628
629
629
630
631
632
633
634
635
635
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
14
```

## Příklad - Načítání textového řetězce 3/4

- Příklad vstupu programu `clang read.c -o read`.

- Vstup soubor `read-in-1.txt`.

```
./read <read_in-1.txt; echo $?
Input string size 11
0
0
hexdump -C read_in-1.txt
00000000 49 20 6c 69 6b 65 20 70 72 70 21 0a
0000000c |I like prp!..|
          lec05/read_in-1.txt
```

- Vstup soubor `read-in-2.txt`.

```
./read <read_in-2.txt; echo $?
ERROR: read return 100
100
0
hexdump -C read_in-2.txt
00000000 49 20 6c 69 6b 65 20 70 72 70 21
0000000b |I like prp!|
          lec05/read_in-2.txt
```

## Část II

### Část 2 – Paměťové třídy, model výpočtu

## Příklad - Načítání textového řetězce 4/4

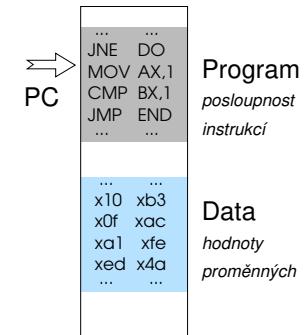
- Generování náhodného vstupu.

```
cat /dev/urandom | env LC_ALL=C tr -dc 'a-zA-Z0-9' | fold -w 10485760 | head -n 1
          lec05/create_rand_string.sh
```

- Omezení paměti programu.

```
clang read.c -o read
./create_rand_string.sh >10MB.txt
du -h 10MB.txt
10M 10MB.txt
./read <10MB.txt
Input string size 10485760
          lec05/read.c
```

## PAMĚŤ



### Paměť počítače s uloženým programem v operační paměti

- Posloupnost instrukcí je čtena z operační paměti.

- Flexibilita ve tvorbě posloupnosti.

*Program lze libovolně měnit.*

- Architektura počítače se společnou pamětí pro data a program.

- von Neumannova architektura počítače

*John von Neumann (1903–1957)*

- sdílí program i data ve stejné paměti.

- Adresa aktuálně prováděné instrukce je uložena v tzv. čítači instrukcí (Program Counter PC).

- Mimoto architektura se sdílenou pamětí umožňuje, aby hodnota ukazatele odkazovala nejen na data, ale také například na část paměti, kde je uložen program (funkce).

*Princip ukazatele na funkci.*

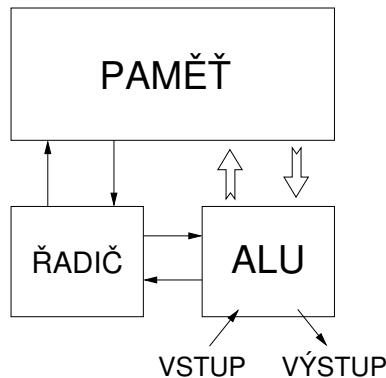
## von Neumannova architektura

V dřívějších případech je program posloupnost instrukcí zpracovávající jednu nebo dvě hodnoty (uložené na nějakém paměťovém místě) jako vstup a generování nějaké výstupní hodnoty, kterou ukládá někam do paměti nebo modifikuje hodnotu PC (podmíněné řízení běhu programu).

- ALU - Aritmeticko logická jednotka (Arithmetic Logic Unit)

Základní matematické a logické instrukce

- PC obsahuje adresu kódu – při volání funkce tak jeho hodnotu můžeme uložit (na zásobník) a následně použít pro návrat na původní místo volání.



## Rozsah platnosti (scope) lokální proměnné

- Lokální proměnné mají rozsah platnosti pouze uvnitř bloku a funkce.

```

1 int a = 1; // globální proměnná
2 void function(void)
3 { // zde a ještě reprezentuje globální proměnnou
4     int a = 10; // lokální proměnná, zastíní globální a
5     if (a == 10) {
6         int a = 1; // nová lokální proměnná a; přístup
7             // na původní lokální a je zastíněn
8         int b = 20; // lokální proměnná s platností pouze
9             // uvnitř bloku
10        a += b + 10; // proměnná a má hodnotu 31
11    } // konec bloku
12    // zde má a hodnotu 10, je to lokální proměnná z řádku 5
13    b = 10; // b není platnou proměnnou
14 }

```

- Globální proměnné mají rozsah platnosti „kdekoli“ v programu.

- Zastíněný přístup lze řešit modifikátorem **extern** (v novém bloku).

[http://www.tutorialspoint.com/cprogramming/c\\_scope\\_rules.htm](http://www.tutorialspoint.com/cprogramming/c_scope_rules.htm)

## Základní rozdělení paměti

- Přidělenou paměť programu můžeme kategorizovat na 5 částí.

- **Zásobník** – lokální proměnné, argumenty funkcií, návratová hodnota funkce.

Spravováno automaticky.

- **Halda** – dynamická paměť (`malloc()`, `free()`).

Spravuje programátor.

- **Statická** – globální nebo „lokální“ `static` proměnné.

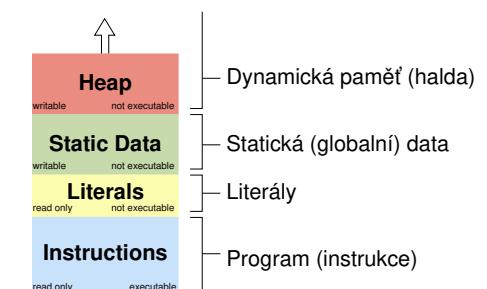
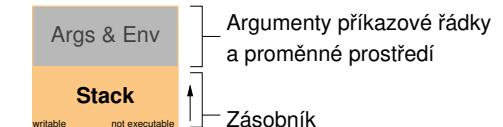
Inicializováno při startu.

- **Literály** – hodnoty zapsané ve zdrojovém kódu programu, např. textové řetězce.

Inicializováno při startu.

- **Program** – strojové instrukce.

Inicializováno při startu.



## Definice vs. deklarace proměnné – extern

- **Definice** proměnné je přidělení paměťového místa proměnné (dle typu). Může být pouze jedna!

- **Deklarace** „oznamuje“, že je proměnná někde definována.

```

1 // extern int global_variable = 10; /* extern
2                         variable with initialization is a
3                         definition */
4 int global_variable = 10;
5 void function(int p);           lec05/extern_var.h
6
7 #include <stdio.h>
8 #include "extern_var.h"
9 static int module_variable;
10 void function(int p)
11 {
12     fprintf(stdout, "function: p %d global
13         variable %d\n", p, global_variable);
14 }                               lec05/extern_var.c

```

```

1 #include <stdio.h>
2 #include "extern_var.h"
3 int main(int argc, char *argv[])
4 {
5     global_variable += 1;
6     function();
7     function();
8     global_variable += 1;
9     function();
10    return 0;
11 }                                lec05/extern-main.c

```

- Vícenásobná definice končí chybou.

```

clang extern_var.c extern-main.c
/tmp/extern-main-619051.:(.data+0x0): multiple
definition of 'global_variable'
/tmp/extern_var-24da84.:(.data+0x0): first
defined here
clang: error: linker command failed with exit
code 1 (use -v to see invocation)

```

## Přidělování paměti proměnným

- **Přidělením paměti proměnné** rozumíme určení paměťového místa pro uložení hodnoty proměnné (příslušného typu) v paměti počítače.
- **Lokálním proměnným** a parametrům funkce se paměť přiděluje při volání funkce.
  - Paměť zůstane přidělena jen do návratu z funkce.
  - Paměť se automaticky alokuje z rezervovaného místa – **zá sobník (stack)**.  
*Při návratu funkce se přidělené paměťové místo uvolní pro další použití.*
  - Výjimku tvoří lokální proměnné s modifikátorem **static**.
    - Z hlediska platnosti rozsahu mají charakter lokálních proměnných.
    - Jejich hodnota je však zachována i po skončení funkce / bloku.
    - Jsou umístěny ve statické části paměti.
- Dynamické přidělování paměti
  - Alokace paměti se provádí funkcí **malloc()**.  
*Nebo její alternativou podle použité knihovny pro správu paměti (např. s garbage collectorem – boehm-gc).*
  - Paměť se alokuje z rezervovaného místa – **halda (heap)**.

Jan Faigl, 2023

B0B36PRP – Přednáška 05: Paměťové třídy

33 / 48

## Příklad rekurzivního volání funkce

- Vyzkoušejte si program pro omezenou velikost zásobníku.

```

1 #include <stdio.h>
3 void printValue(int v)
4 {
5     printf("value: %i\n", v);
6     printValue(v + 1);
7 }
9 int main(void)
10 {
11     printValue(1);
12 }

    lec05/demo-stack_overflow.c

```

```

clang demo-stack_overflow.c
ulimit -s 10000; ./a.out | tail -n 3
value: 319816
value: 319817
Segmentation fault
ulimit -s 1000; ./a.out | tail -n 3
value: 31730
value: 31731
Segmentation fault

```

Jan Faigl, 2023

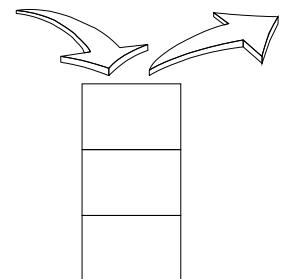
B0B36PRP – Přednáška 05: Paměťové třídy

35 / 48

## Zásobník

- Úseky paměti přidělované lokálním proměnným a parametrům funkce tvoří tzv. **zá sobník (stack)**.
- Úseky se přidávají a odebírají.
  - Vždy se odebere naposledy přidaný úsek.

*LIFO – last in, first out.*
- Na zásobník se ukládá „volání funkce“.
- Ze zásobníku se alokují proměnné parametrů funkce.



*Na zásobník se také ukládá návratová hodnota funkce a také hodnota „program counter“ původně prováděné instrukce, před voláním funkce.*

*Opakováním rekurzivním volání funkce můžeme zaplnit velikost přiděleného zásobníku a program skončí chybou.*

Jan Faigl, 2023

B0B36PRP – Přednáška 05: Paměťové třídy

34 / 48

## Návratová hodnota funkce a kódovací styl **return** 1/2

- Předání hodnoty volání funkce je předepsáno voláním **return**.

```

int doSomethingUseful() {
    int ret = -1;
    ...
    return ret;
}

```

- Jak často umisťovat volání **return** ve funkci?

```

int doSomething() {
    if (
        !cond1
        && cond2
        && cond3
    ) {
        ... do some long code ...
    }
    return 0;
}

int doSomething() {
    if (cond1) {
        return 0;
    }
    if (!cond2) {
        return 0;
    }
    if (!cond3) {
        return 0;
    }
    ... some long code ....
    return 0;
}

```

<http://llvm.org/docs/CodingStandards.html>

Jan Faigl, 2023

B0B36PRP – Přednáška 05: Paměťové třídy

36 / 48

## Návratová hodnota funkce a kódovací styl **return** 2/2

- Volání **return** na začátku funkce může být přehlednější.
- Podle hodnoty podmínky je volání funkce ukončeno.
- Kódovací konvence může také předepisovat použití nejvýše jednoho volání **return**.  
Má výhodu v jednoznačné identifikaci místa volání, můžeme pak například jednoduše přidat další zpracování výstupní hodnoty funkce.
- Dále není doporučováno bezprostředně používat **else** za voláním **return** (nebo jiným přerušením toku programu), např.

```
case 10:
    if (...) {
        ...
        return 1;
    } else {
        if (cond) {
            ...
            return -1;
        } else {
            break;
        }
    }
```

```
case 10:
    if (...) {
        ...
        return 1;
    } else {
        if (cond) {
            ...
            return -1;
        }
    }
break;
```

Jan Faigl, 2023

B0B36PRP – Přednáška 05: Paměťové třídy

37 / 48

## Proměnné – paměťová třída

- Specifikátory paměťové třídy (Storage Class Specifiers – SCS).
  - **auto** (lokální) – Definuje proměnnou jako dočasnou (automatickou). Lze použít pro lokální proměnné definované uvnitř funkce. Jedná se o implicitní nastavení, platnost proměnné je omezena na blok. Proměnná je v **zá sobníku**.
  - **register** – Doporučuje překladač umístit proměnnou do registru procesoru (rychlosť přístupu). Překladač může, ale nemusí vyhovět. Jinak stejně jako **auto**.  
*Zpravidla řešíme překladem s optimalizacemi.*
  - **static**
    - Uvnitř bloku **{...}** – definujeme proměnnou jako statickou, která si **ponechává hodnotu i při opuštění bloku**. Existuje po celou dobu chodu programu. Je uložena v **datové oblasti**.
    - Vně bloku – kde je implicitně proměnná uložena v **datové oblasti** (statická) omezuje její viditelnost na modul.
  - **extern** – rozšiřuje viditelnost statických proměnných z modulu na celý program. Globální proměnné s **extern** jsou definované v **datové oblasti**.

Jan Faigl, 2023

B0B36PRP – Přednáška 05: Paměťové třídy

40 / 48

## Proměnné

- Proměnné představují vymezenou oblast paměti a v C je můžeme rozdělit podle způsobu alokace.
  - **Statická** alokace – provede se při definici **statické** nebo globální proměnné; paměťový prostor je alokován při startu programu a nikdy není uvolněn.
  - **Automatická** alokace – probíhá automaticky v případě lokálních proměnných (nebo argumentů funkce); paměťový prostor je alokován na **zá sobníku** a paměť proměnné je automaticky uvolněna s koncem platnosti proměnné.
- Např. po ukončení bloku funkce.
- **Dynamická** alokace – není podporována přímo jazykem C, ale je přístupná knihovními funkcemi.

Např. **malloc()** a **free()** z knihovny `<stdlib.h>` nebo `<malloc.h>`  
[http://gribblelab.org/CBootcamp/7\\_Memory\\_Stack\\_vs\\_Heap.html](http://gribblelab.org/CBootcamp/7_Memory_Stack_vs_Heap.html)

Jan Faigl, 2023

B0B36PRP – Přednáška 05: Paměťové třídy

39 / 48

## Příklad definice proměnných

- Hlavničkový soubor **vardec.h**

```
1 extern int global_variable;
```
- Zdrojový soubor **vardec.c**

```
1 #include <stdio.h>
2 #include "vardec.h"
4 static int module_variable;
5 int global_variable;
7 void function(int p);
9 int main(void)
10 {
11     int local;
12     function(1);
13     function(1);
14     function(1);
15     return 0;
16 }
```
- Výstup

```
1 func: p 1, lv 1, slv 1
2 func: p 1, lv 1, slv 2
3 func: p 1, lv 1, slv 3
```

Uvedený příklad demonstreuje různé definice proměnných. V případě proměnné **global\_variable** je její definice v modulu s funkci **main()** diskutabilní. Modul **vardec.o** nebude linkovat s jiným program s vlastní (jinou) funkci **main()**.

Jan Faigl, 2023

B0B36PRP – Přednáška 05: Paměťové třídy

41 / 48

## Definice proměnných a operátor přiřazení

- Proměnné definujeme uvedením typu a jména proměnné.
  - Jména proměnných volíme malá písmena.
  - Víceslová jména zapisujeme s podtržkem `_` nebo volíme tzv. *camelCase*.
- <https://en.wikipedia.org/wiki/CamelCase>
- Proměnné definujeme na samostatném řádku.

```
1 int n;
2 int number_of_items;
```

- Příkaz přiřazení se skládá z operátoru přiřazení `=` a ;
  - Levá strana přiřazení musí být **l-value – location-value, left-value** – musí reprezentovat paměťové místo pro uložení výsledku.
  - Přiřazení je výraz a můžeme jej tak použít všude, kde je dovolen výraz příslušného typu.

```
1 /* int c, i, j; */
2 i = j = 10;
3 if ((c = 5) == 5) {
4     fprintf(stdout, "c is 5 \n");
5 } else {
6     fprintf(stdout, "c is not 5\n");
7 }
```

lec05/assign.c

## Část III

### Část 3 – Zadání 5. domácího úkolu (HW05)

## Zadání 5. domácího úkolu HW05

### Téma: Caesarova šifra

Povinné zadání: **3b**; Volitelné zadání: **2b**; Bonusové zadání: **není**

- **Motivace:** Získat zkušenosti s dynamickou alokací paměti. Implementovat výpočetní úlohu optimalizačního typu.
- **Cíl:** Osvojit si práci s dynamickou alokací paměti.
- **Zadání:** <https://cw.fel.cvut.cz/wiki/courses/b0b36prp/hw/hw05>
  - Načtení dvou vstupních textů a tisk dekódované zprávy na výstup.
  - Zakódovaný text i (špatně) odposlechnutý text mají stejně délky.
  - Nalezení největší shody dekódovaného a odposlechnutého textu na základě hodnoty posunu v Caesarově šifře.
  - Optimalizace hodnoty Hammingovy vzdálenosti.
- [https://en.wikipedia.org/wiki/Hamming\\_distance](https://en.wikipedia.org/wiki/Hamming_distance)
- **Volitelné zadání** rozšiřuje úlohu o uvažování chybějících znaků v odposlechnutém textu, což vede na využití Levenshteinovy vzdálenosti.
- [https://en.wikipedia.org/wiki/Levenshtein\\_distance](https://en.wikipedia.org/wiki/Levenshtein_distance)
- **Termín odevzdání:** **25.11.2023, 23:59:59 PST.**

### Shrnutí přednášky

## Diskutovaná téma

- Ukazatele a modifikátor `const`
- Dynamická alokace paměti
- Ukazatel na funkce
- Paměťové třídy
- Volání funkcí
  
- Příště: Struktury a union, přesnost výpočtu a vnitřní reprezentace číselných typů.

Jan Faigl, 2023

B0B36PRP – Přednáška 05: Paměťové třídy

46 / 48

Kódovací příklad – NATO Abeceda

Kódovací příklad – NATO Abeceda („jinak“)

Kódovací příklad – Rotace textového řetězce

## Kódovací příklad – NATO Abeceda – 1/4

- Implementujme program, který převede vstupní text (ASCII, znaky A–Z a a–z) do NATO abecedy, ve které jsou písmena hláskována prostřednictvím následujících jmen.

  - Alpha, Bravo, Charlie, Delta, Echo, Foxtrot, Golf, Hotel, India, Juliett, Kilo, Lima, Mike, November, Oscar, Papa, Quebec, Romeo, Sierra, Tango, Uniform, Victor, Whiskey, X-ray, Yankee, Zulu.

- V programu definujeme pole ukazatelů na textové literály s jednotlivými slovy.
- Programově otestujeme, že slova odpovídají počátečním písmenům A–Z.

■ Očekávaný výstup pro vstup `in.txt`.

```
$ cat in.txt
I like PRP and programming in C.
$ clang nato-alphabet.c && ./a.out < in.txt 2>/dev/null
India Lima India Kilo Echo Papa Romeo Papa Alpha
November Delta Papa Romeo Oscar Golf Romeo Alpha
Mike Mike India November Golf India November
Charlie
```

## ■ Implementujeme testovací funkce.

```
1 static char *words[] = { // static to be "private"
2     "Alpha", "Bravo", "Charlie", "Delta", "Echo", "
3     "Foxtrot", "Golf", "Hotel", "India", "Juliett", "
4     "Kilo", "Lima", "Mike", "November", "Oscar", "Papa",
5     "Quebec", "Romeo", "Sierra", "Tango", "Uniform", "
6     "Victor", "Whiskey", "X-ray", "Yankee", "Zulu", NULL
7 }; // it is an array of pointers to text literals
8 int count_words_array(char *words[]); // Using array
9 int count_words(char **words); // Pointer to pointers
10 bool check_alphabet_words(char *words[]);
```

Kódovací příklad – NATO Abeceda

Kódovací příklad – NATO Abeceda („jinak“)

Kódovací příklad – Rotace textového řetězce

Část V  
Appendix

Jan Faigl, 2023

B0B36PRP – Přednáška 05: Paměťové třídy

47 / 48

Kódovací příklad – NATO Abeceda

Kódovací příklad – NATO Abeceda („jinak“)

Kódovací příklad – Rotace textového řetězce

## Kódovací příklad – NATO Abeceda – 2/4

```
1 // array is terminated by NULL used for counting
2 static char *words[] = { "Alpha", ... , "Zulu", NULL };
3 // array-like variant
4 int count_words_array(char *words[])
5 {
6     int n = 0;
7     while(words[n] != NULL) {
8         fprintf(stderr, "DEBUG: \"%s\\n\", words[n]);
9         n += 1;
10    }
11    return n;
12 }
13 // pure pointer variant
14 int count_words(char **words)
15 {
16     int n = 0;
17     char **cur = words;
18     while (*cur) {
19         fprintf(stderr, "DEBUG: \"%s\\n\", *cur);
20         cur++;
21         n += 1;
22     }
23     return n;
24 }
25
26 bool check_alphabet_words(char *words[])
27 {
28     bool ret = true; // true is from #include <stdbool.h>
29     char c = 'A'; // char is an integer ASCII code number
30     char **cur = &words[0]; // there is always at least one item
31     while (*cur) {
32         fprintf(stderr, "DEBUG: check %s[0] for '%c'\\n", *cur, c);
33         if (c != *cur[0]) { // the first letter needs to match
34             ret = false; // false is from #include <stdbool.h>
35             break;
36         } else {
37             c += 1;
38             cur += 1;
39         }
40     }
41     return ret;
42 }
```

- Pole `words` je posloupnost prvků stejného typu (ukazatel na `char` – textový řetězec).
- Hodnota `&words[0]` je identická adresa jako hodnota `words`.

Jan Faigl, 2023

B0B36PRP – Přednáška 05: Paměťové třídy

49 / 48

Jan Faigl, 2023

B0B36PRP – Přednáška 05: Paměťové třídy

50 / 48

## Kódovací příklad – NATO Abeceda – 3/4

- Můžeme použít `const`.

```

1 static const char * const words[] = { "Alpha", ..., NULL };
2 int count_words_array(const char * const words[])
3 {
4     int n = 0;
5     while(words[n] != NULL) {
6         n += 1;
7     }
8     return n;
9 }
10 int count_words(const char * const * const words)
11 {
12     int n = 0;
13     // cur je ukazatel na data typu konstantní
14     // ukazatel na konstantní textový řetězec
15     // (na konstantní ukazatel na konstantní hodnoty char).
16     const char * const * cur = words; // cur chceme měnit
17     while (*cur) {
18         cur++; // cur nemí konstantní ukazatel
19         n += 1;
20     }
21     return n;
22 }
23

```

```

26 #include <stdio.h>
27 #include <stdbool.h>
28
29 static const char * const words[] = { "Alpha", ..., NULL };
30
31 int count_words_array(const char * const words[]);
32
33 int count_words(const char * const * const words);
34
35 bool check_alphabet_words(const char * const words[]);
36
37 int main(void)
38 {
39     int ret = EXIT_SUCCESS;
40     fprintf(stderr, "DEBUG: size %lu\n", sizeof(words));
41     int n = count_words_array(words);
42     fprintf(stderr, "DEBUG: no. of words: %i\n", n);
43     n = count_words(&words[0]);
44     fprintf(stderr, "DEBUG: no. of words: %i\n", n);
45     bool checked = check_alphabet_words(words);
46     if (!checked) {
47         fprintf(stderr, "DEBUG: check_alphabet_words passed [%s]\n",
48             checked ? "OK" : "FAIL");
49     }
50     return ret;
51 }

```

Jan Faigl, 2023

B0B36PRP – Přednáška 05: Paměťové třídy

51 / 48

## Kódovací příklad – NATO Abeceda („jinak“) – 1/2

- Slova abecedy uložíme jako řetězec `alphabet` všech slov spojených bez mezery, do kterého budeme odkazovat na jednotlivá slova polem ukazatelů na textové řetězce (`words`).

- Slov je 'Z' - 'A' + 1, ale řetězec je posloupnost znaků zakončená '\0'.
- První písmeno slova abecedy používáme k indexaci, např. 'C'Charlie je odkazované ukazatelem `words['C' - 'A']`. První znak slova tak můžeme v abecedě `alphabet` nahradit znakem '\0' získáme textové řetězce.

Bez prvního znaku!

```

1 //Ukazatel na textový literál. Literál nemůžeme měnit!
2 //static char *alphabet = "AlphaBravoCharlie...";
3 static char alphabet[] =
4     "AlphaBravoCharlieDeltaEchoFoxtrotGolfHotelIndia"
5     "JuliettKiloLimaMikeNovemberOscarPapaQuebecRomeo"
6     "SierraTangoUniformVictorWhiskeyX-rayYankeeZulu";
7 //pole ukazatelů na textové řetězce
8 static char *words['Z' - 'A' + 1] = { [0] = NULL };
9
10 int fill_words(char* str, char *words[]);
11
12 int main(void)
13 {
14     int ret = fill_words(alphabet, words);
15     if (!ret) {
16         for (char c = 'A'; c <= 'Z'; ++c) {
17             fprintf(stderr, "DEBUG: %02d. '%c' - %c%\n",
18                 c, c, words[c - 'A']);
19         }
20     }
21     return ret;
22 }

```

```

24 int fill_words(char* alphabet, char *words[])
25 {
26     int ret = EXIT_SUCCESS;
27     char *cur = alphabet; // kurzor do pole s písmeny abecedy
28     for (char c = 'A'; c <= 'Z'; ++c) {
29         assert(words[c - 'A'] == NULL); // nemá být nastaveno
30         cur = strchr(cur, c); // vyhledání řetězce začínající c
31         assert(*cur); // písmeno c musí být v abecedě
32         *cur = '\0';
33         words[c - 'A'] = ++cur; // nastavení a posun kurzoru
34         assert(words[c - 'A']); // it should be set now
35     }
36     return ret; // pragmaticky vždy EXIT_SUCCESS nebo assert.
37 }

```

- V implementaci použijeme (makro) `assert()` k testování správné inicializace datových struktur.

Makro slouží pro ladění, viz man assert.

Jan Faigl, 2023

B0B36PRP – Přednáška 05: Paměťové třídy

54 / 48

## Kódovací příklad – NATO Abeceda – 4/4

```

1 ...
2 static const char * const words[] = { "Alpha", ..., NULL };
3 ...
4 char my_toupper(char c);
5
6 int main(void)
7 {
8     ...
9     int c;
10    while ((c = getchar()) != EOF) {
11        c = my_toupper(c); // or toupper() from <ctype.h>
12        if (c >= 'A' && c <= 'Z') {
13            printf("%s ", words[c - 'A']); // always print space
14        }
15    }
16    ...
17 }
18
19 char my_toupper(char c) // or use toupper() from <ctype.h>
20 {
21     if (c >= 'a' && c <= 'z') {
22         c = c - 'a' + 'A';
23     }
24     return c;
25 }

```

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <assert.h>
4 #include <stdbool.h>
5
6 static const char * const words[] = { "Alpha", ..., NULL };
7 ...
8 int count_words_array(const char * const words[]);
9 bool check_alphabet_words(const char * const words[]);
10
11 int main(void)
12 {
13     // assert macro debug and development only, see -DNDEBUG
14     assert(count_words_array(words) == 'Z' - 'A' + 1);
15     assert(check_alphabet_words(words));
16     int c;
17     while ((c = getchar()) != EOF) {
18         c = (c >= 'a' && c <= 'z') ? c - 'a' + 'A' : c;
19         if (c >= 'A' && c <= 'Z') {
20             printf("%s\n", words[c - 'A']);
21         }
22     }
23     return EXIT_SUCCESS;
24 }

```

- V rámci zpřehlednění můžeme překlad (řádky 15–21) dát do samostatné funkce `void translate(const char * const words[])`.

Jan Faigl, 2023 B0B36PRP – Přednáška 05: Paměťové třídy 52 / 48

## Kódovací příklad – NATO Abeceda („jinak“) – 2/2

- Přidáme překlad znaků načítaných ze `stdin` a implementaci zpřehlednění.

```

1 static char alphabet[] =
2     "AlphaBravoCharlieDeltaEchoFoxtrotGolfHotelIndia"
3     "JuliettKiloLimaMikeNovemberOscarPapaQuebecRomeo"
4     "SierraTangoUniformVictorWhiskeyX-rayYankeeZulu";
5 static char *words['Z' - 'A' + 1] = { [0] = NULL };
6
7 int fill_words(char* str, char *words[]);
8
9 int main(void)
10 {
11     int ret = fill_words(alphabet, words);
12     if (!ret) {
13         for (char c = 'A'; c <= 'Z'; ++c) {
14             fprintf(stderr, "DEBUG: %02d. '%c' - %c%\n",
15                 c, c, words[c - 'A']);
16         }
17     }
18     int c;
19     while ((c = getchar()) != EOF) {
20         c = (c >= 'a' && c <= 'z') ? c - 'a' + 'A' : c;
21         // volání funkce toupper() bude zpřehledněši!
22         if (c >= 'A' && c <= 'Z') {
23             printf("%c ", words[c - 'A']);
24         }
25     }
26     return ret;
27 }

```

```

1 static char alphabet[] =
2     "AlphaBravoCharlieDeltaEchoFoxtrotGolfHotelIndia"
3     "JuliettKiloLimaMikeNovemberOscarPapaQuebecRomeo"
4     "SierraTangoUniformVictorWhiskeyX-rayYankeeZulu";
5 static char *words['Z' - 'A' + 1] = { [0] = NULL };
6
7 void fill_words(char* str, char *words[]);
8 void translate(char *words[]);
9
10 int main(void)
11 {
12     fill_words(alphabet, words);
13     translate(words);
14     return EXIT_SUCCESS;
15 }
16
17 void translate(char *words[])
18 {
19     int c;
20     while ((c = getchar()) != EOF) {
21         c = toupper(c); // funkce z #include<ctype.h>
22         if (c >= 'A' && c <= 'Z') {
23             printf("%c ", c, words[c - 'A']); // první znak!
24         }
25     }
26 }

```

- Další rozšíření programu může být zpracování jiných znaků, než znaků abecedy 'A'-'Z' a 'a'-'z'.

Jan Faigl, 2023 B0B36PRP – Přednáška 05: Paměťové třídy 55 / 48

## Kódovací příklad – Rotace textového řetězce – 1/4

- Implementujme program, který načte ze `stdin` dva textové řetězce (dva řádky zakončené `\n`) a pokuší se najít rotaci (posunutí – `offset`) druhého řádku tak, aby odpovídala prvnímu řádku.
- Oba řádky (řetězce) předpokládáme, že jsou stejně dlouhé.
- Chybu dynamické alokace program indikuje návratovou hodnotou `129`, chybu vstupu hodnotou `100`, jinak vrací `EXIT_SUCCESS`.
- Délka řetězců je až do maximálního hodnoty `size_t`, posunutí pouze do `INT_MAX`.
- V případě neúspěšné dynamické alokace program ukončujeme voláním `exit(129)`;

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <limits.h> // for INT_MAX
5 #ifndef INIT_LEN
6 #define INIT_LEN 8
7 #endif
8
9 enum { ERROR_OK = EXIT_SUCCESS, ERROR_IN = 100, ERROR_MEM = 129 };
10 void* my_realloc(void *ptr, size_t size,
11                  const char *file, const int line);
12
13 void* my_realloc(void *ptr, size_t size,
14                  const char *file, const int line)
15 {
16     void* ret = realloc(ptr, size);
17     if (!ret) {
18         fprintf(stderr, "ERROR: Cannot realloc %lu bytes -- called
19             at %s:%i\n", size, file, line);
20         free(ptr);
21         exit(ERROR_MEM);
22     }
23     return ret;
24 }
25
26     ■ Volání realloc() alokuje nebo přealokuje paměť.
27     ■ Funkci předáváme soubor a číslo řádku, kde funkci my_realloc()
28     voláme, pro indikaci, kde došlo k chybě.
29 }
```

Jan Faigl, 2023

B0B36PRP – Přednáška 05: Paměťové třídy

49 / 48

## Kódovací příklad – Rotace textového řetězce – 3/4

```

1 char* read_line(void)
2 {
3     size_t capacity = INIT_LEN;
4     char *str = my_realloc(NULL, sizeof(char) * (INIT_LEN + 1),
5                           __FILE__, __LINE__); // +1 for '\0'
6     size_t len = 0;
7     int c;
8     while ((c = getchar()) != EOF && c != '\n') {
9         if (len == capacity) {
10             capacity *= 2;
11             str = my_realloc(str, sizeof(char) * (capacity + 1),
12                               __FILE__, __LINE__); // +1 for '\0'
13         }
14         str[len++] = c;
15     }
16     if (len > 0) {
17         str[len] = '\0';
18     } else {
19         free(str);
20         str = NULL;
21     }
22     return str;
23 }
24
25     ■ read_line() vrací NULL pouze pokud je načten prázdný řádek.
26     ■ Chyba alokace dynamické paměti ukončí program voláním exit()
27     v naší funkci my_realloc().
28 }
```

```

1 char* shift(int offset, const char* src, size_t n, char *dst)
2 {
3     for (size_t i = 0; i < n; ++i) { // n type is size_t !!!
4         dst[i] = src[(offset + i) % n];
5     }
6     return dst;
7 }
8
9 int get_offset(const char *s1, size_t n1, const char *s2, size_t n2)
10 { // we already checked that s1 && s2 && n1 == n2
11     int ret = -1;
12     int max_shift = INT_MAX < n2 ? INT_MAX : n2; // limits.h
13     char *s = my_realloc(NULL, sizeof(char) * (n2 + 1), __FILE__,
14                           __LINE__); // +1 for '\0'
15     for (int i = 0; i < max_shift; ++i) {
16         s = shift(i, s2, n2, s); // shift s2 to s and return s
17         if (strcmp(s1, s) == 0) { // strings matched
18             ret = i; // perfect match, exit the loop
19             break;
20         }
21     }
22     free(s); // s is dynamically allocated, release the memory
23     return ret;
24 }
25
26     ■ Posuneme 2. řádek (s) a testujeme jestli je identický s 1. řádkem.
27     ■ Funkce strcmp() porovnává řetězce lexicograficky, proto vrací int.
28 }
```

Jan Faigl, 2023

B0B36PRP – Přednáška 05: Paměťové třídy

51 / 48

## Kódovací příklad – Rotace textového řetězce – 2/4

```

14 char* read_line(void); // read a line from stdin, terminated by '\n' return as null-terminated string
15 char* shift(int offset, const char* src, size_t n, char *dst); // src and dst are strings at least n long (+1 for '\0')
16 int get_offset(const char *s1, size_t n1, const char *s2, size_t n2); // offset - max INT_MAX; strings - up to can size_t
17 int print_offset(const char *s, size_t n, int offset);
18 int main(void)
19 {
20     int ret = ERROR_OK;
21     char *l1 = read_line();
22     char *l2 = read_line();
23     size_t n1, n2;
24
25     if (l1 && l2 && (n1 = strlen(l1)) == (n2 = strlen(l2))) {
26         printf(stderr, "DEBUG: l1[%lu]: \"%s\\n\", n1, l1);
27         printf(stderr, "DEBUG: l2[%lu]: \"%s\\n\", n2, l2);
28         int offset = get_offset(l1, n1, l2, n2);
29         fprintf(stdout, "Matching offset %d\\n", offset);
30         offset >= 0 && print_offset(l2, n2, offset); // call print_offset only if offset >= 0
31     } else {
32         fprintf(stderr, "ERROR: Wrong input!\\n");
33         ret = ERROR_IN;
34     }
35     free(l1); // free(ptr) - If ptr is NULL no action occurs.
36     free(l2); // See man free.
37     return ret;
38 }
39
40     ■ free(l1); // free(ptr) - If ptr is NULL no action occurs.
41     ■ free(l2); // See man free.
42 }
```

Jan Faigl, 2023

B0B36PRP – Přednáška 05: Paměťové třídy

50 / 48

## Kódovací příklad – Rotace textového řetězce – 4/4

■ K vytisknutí posunutého řetězce v samostatné funkci `print_offset()` alokujeme dynamickou paměť, kterou před ukončení funkce opět uvolníme.

```

105 int print_offset(const char *s, size_t n, int offset)
106 {
107     int ret = 1;
108     char *str = my_realloc(NULL, sizeof(char) * (n + 1),
109                           __FILE__, __LINE__); // +1 for '\0'
110     shift(offset, s, n, str);
111     fprintf(stderr, "DEBUG: shift: \"%s\\n\", str");
112     free(str);
113     return ret;
114 }
```

```

106     ■ char *l1 = read_line();
107     ■ char *l2 = read_line();
108     ■ size_t n1, n2;
109     ■ if (l1 && l2 && (n1 = strlen(l1)) == (n2 = strlen(l2))) {
110         ■ printf(stderr, "DEBUG: l1[%lu]: \"%s\\n\", n1, l1);
111         ■ printf(stderr, "DEBUG: l2[%lu]: \"%s\\n\", n2, l2);
112         ■ int offset = get_offset(l1, n1, l2, n2);
113         ■ fprintf(stdout, "Matching offset %d\\n", offset);
114         ■ offset >= 0 && print_offset(l2, n2, offset);
115     } else {
116         ■ fprintf(stderr, "ERROR: Wrong input!\\n");
117         ■ ret = ERROR_IN;
118     }
119 }
```

Jan Faigl, 2023

B0B36PRP – Přednáška 05: Paměťové třídy

52 / 48

## ■ Program testujeme pro ukázkový vstup.

```

1 Lorem ipsum dolor sit amet.
2 sit amet.Lorem ipsum dolor
3 
```

```

4 $ clang -g shift.c -o shift && ./shift <input.txt; echo $?
5 DEBUG: 11[27]: "Lorem ipsum dolor sit amet."
6 DEBUG: 12[27]: "sit amet.Lorem ipsum dolor "
7 Matching offset 9
8 DEBUG: shift: "Lorem ipsum dolor sit amet."
9 0
10 
```

■ Vyzkoušejte si chování programu v kombinaci s `valgrind` pro detekci chybného přístupu k paměti, např. chybňá alokace paměti pro posunutý řetězec.

```

11 for (size_t i = 0; i < n; ++i) {
12     dst[i] = src[(offset + i) % n];
13 }
14
15 $ valgrind ./shift < input.txt
16 ...
17 ==80708== Invalid write of size 1
18 ==80708== at 0x202240: shift (shift.c:84)
19 ==80708== by 0x202092: get_offset (shift.c:95)
20 ==80708== by 0x201DF2: main (shift.c:36)
21 
```