C - More on pointers, trees, linked lists

Programming Expertise - session 05

Christian Kappel

AG Genetik
Universität Potsdam

16 May 2024

Outline

Introduction

C sessions

- Intro-Kurs C/C++
- Data Types, Control Constructs
- Functions
- Arrays and Pointers
- **More on Pointers, Trees, Linked Lists**
- File input/output

Pointer

```c
#include <stdio.h>

int main() {
  // Initilize variable i
  int i = 99;
  // Set pointer to variable i
  int *pointer_i = &i;

  // i and *pointer_i refer to value
  printf("i is %d\n", i);
  printf("*pointer_i is %d\n", *pointer_i);
  // &i and pointer_i refer to address
  printf("%p, %p\n", &i, pointer_i);

  return(0);
}
```

Arrays are passed as pointers

```c
#include <stdio.h>

int main()
{
  int a[5] = {1,2,3,4,5};

  printf("a is %p.\n", a); // address of first element
  printf("a[0] is %d.\n", a[0]); // 1
  printf("*a is %d.\n", *a); // 1

  return(0);
}
```

Arrays are passed as pointers

```c
#include <stdio.h>
void fun(int array[]);

int main()
{
  int i,a[5] = {1,2,3,4,5};
  for(i=0; i<sizeof(a)/sizeof(int); i++)
    printf("Entry %d is %d.\n", i+1, a[i]);
  fun(a);
  return(0);
}

void fun(int array[])
{
  printf("sizeof(array)/sizeof(int)=%d\n",sizeof(array)/sizeof(int))
      ;
  /* sizeof(array) giving the size of a pointer, not of the array.
     8 bytes on 64 bit machines. */
}
```

Arrays are passed as pointers

```c
#include <stdio.h>
void fun(int *array); // equals array[]

int main()
{
  int i,a[5] = {1,2,3,4,5};
  for(i=0; i<sizeof(a)/sizeof(int); i++)
    printf("Entry %d is %d.\n", i+1, a[i]);
  fun(a);
  return(0);
}

void fun(int *array)
{
  printf("sizeof(array)/sizeof(int)=%d\n",sizeof(array)/sizeof(int))
      ;
  /* sizeof(array) giving the size of a pointer, not of the array.
     8 bytes on 64 bit machines. */
}
```

Arrays are passed as pointers

```c
#include <stdio.h>

void fun(int array[], int array_size);

int main()
{
  int a[5] = {2,4,6,8,10};
  fun(a,5); // Passing array size
  return(0);
}

void fun(int array[], int array_size)
{
  int i;
  for(i=0; i<array_size; i++)
    {
      printf("Element %d is %d.\n", i+1, *array++);
    }
}
```

Arrays and pointer arithmetics

```c
#include <stdio.h>

int main()
{
  int i, a[5] = {1,2,3,4,5};

  for(i=0; i<5; i++)
    {
      printf("a[%d]=%d\n", i, *a++);
      // Not working. Array addresses cannot be changed.
    }

  return(0);
}
```

Arrays and pointer arithmetics

```c
#include <stdio.h>

int main()
{
  int i, a[5] = {1,2,3,4,5};
  int *pointer_a = a; // pointer that can be modified

  for(i=0; i<5; i++)
    {
      printf("a[%d]=%d\n", i, *pointer_a++);
    }

  return(0);
}
```

Arrays and pointer arithmetics

```c
#include <stdio.h>

int main()
{
  int i, a[5] = {2,4,6,8,10};

  for(i=0; i<5; i++)
    {
      printf("a[%d]=%d\n", i, *(a+i));
      // This works. Array address is not changed.
    }

  return(0);
}
```

More on pointers

Passing by reference

```c
#include <stdio.h>

void swap(int *a, int *b);

int main()
{
  int a = 1;
  int b = 2;
  printf("a=%d, b=%d\n",a,b);
  swap(&a,&b);
  printf("a=%d, b=%d\n",a,b);
  return(0);
}

void swap(int *a, int *b)
{
  int tmp = *b;
  *b = *a;
  *a = tmp;
}
```

Pointers to pointers

```c
#include <stdio.h>

int main()
{
   int i = 23;
   int *p1, **p2;

   p1 = &i;
   p2 = &p1;

   printf("i = %d\n", i); // 23
   printf("p1 = %p\n", p1); // addr1
   printf("*p1 = %d\n", *p1); // 23
   printf("p2 = %p\n", p2); // addr2
   printf("*p2 = %p\n", *p2); // addr1
   printf("**p2 = %d\n", **p2); // 23

   return(0);
}
```

Arrays of pointers

```
1  #include <stdio.h>
2
3  int main()
4  {
5     char *welcome[4] = { "My", "name", "is", "Christian" };
6
7     int i;
8     for(i=0; i<4; i++)
9        printf("%s ", welcome[i]);
10    printf("\n");
11
12    return(0);
13 }
```

Dynamic memory allocation

- malloc() function allows dynamic memory allocation.
- Looks for free memory and returns address, if found.
- Part of stdlib.h

Dynamic memory allocation

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
  char *string; // Allocate memory for a 95 character string
  string = (char*) malloc(96);

  int count;
  char *p = string;
  for(count=32; count<126; count++)
    *p++ = count;
  *p++ = '\0';

  puts(string);
  printf("%s\n", string);

  free(string); // Freeing allocated memory

  return(0);
}
```

Dynamic memory allocation

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
  // Allocate memory for an user defined integer array
  int size;
  scanf("%d", &size);

  int *a;
  a = (int*) malloc(size * sizeof(int));

  free(a)

  return(0);
}
```

Dynamic memory allocation

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
  // Allocate memory for an user defined integer array
  int size;
  scanf("%d", &size);

  int *a = (int*) malloc(size * sizeof(int));
  if(a == NULL)
    {
      puts("Requested memory couldn't be allocated!");
      exit(1);
    }

  free(a);
  return(0);
}
```

Trees and linked lists

Structures

```c
#include <stdio.h>
#include <strings.h>

int main()
{
  struct sample {
    int height;
    int width;
  };

  struct sample s1;

  s1.height = 150;
  s1.width = 180;

  printf("h=%d, w=%d\n", s1.height, s1.width);

  return(0);
}
```

Structures

```c
#include <stdio.h>
#include <strings.h>

int main()
{
   struct sample {
      char *name;
      int height;
      int width;
   };

   struct sample s1;
   s1.name = "Bacteria1";
   s1.height = 150;
   s1.width = 180;

   printf("%s: h=%d, w=%d\n", s1.name, s1.height, s1.width);

   return(0);
}
```

Linked lists

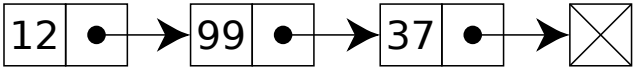

Figure: https://commons.wikimedia.org/wiki/File:Singly-linked-list.svg

Linked lists

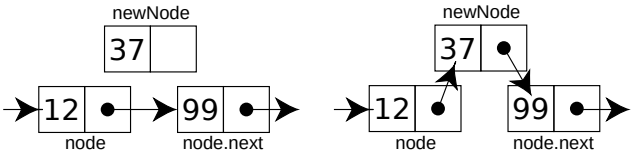

Figure: https://commons.wikimedia.org/wiki/File:CPT-LinkedLists-addingnode.svg

Linked lists

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
  struct student {
    char name[23];
    struct student *next;
  };

  typedef struct student student;

  student *new, *current, *head=NULL;

  // Adding first element
  new = (student*) malloc(sizeof(student));
  strcpy(new->name, "Christian");
  new->next = NULL;
  head = new;
```

Linked lists


```
21
22    // Adding additional element to the end
23    current = head;
24    while(current->next != NULL)
25      {
26        current = current->next;
27      }
28    new = (student*) malloc(sizeof(student));
29    strcpy(new->name, "Michael");
30    new->next = NULL;
31    current->next = new;
```

Linked lists

```
32
33    // Print out elements
34    current = head;
35    while(current != NULL)
36      {
37        printf("%s\n", current->name);
38        current = current->next;
39      }
```

Linked lists

```
40
41    // Free memory
42    while(head != NULL)
43      {
44        current = head;
45        head = head->next;
46        free(current);
47      }
48
49    return(0);
50  }
```

Binary tree

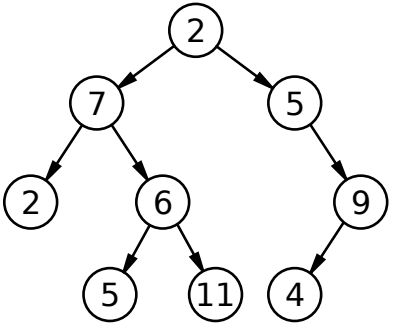

Figure: https://commons.wikimedia.org/wiki/File:Binary_tree.svg

Binary tree


```c
#include <stdio.h>
#include <stdlib.h>

struct node
{
  int value;
  struct node *left;
  struct node *right;
};

int main()
{
  struct node* firstNode = (struct node*) malloc(sizeof(struct node)
      );

  firstNode->value = 0;
  firstNode->left = NULL;
  firstNode->right = NULL;

  return(0);
}
```

Questions

- Thank you very much for your attention.
- Please feel free to ask questions.
- You may write to christian.kappel@uni-potsdam.de for any further or upcoming questions.

References I