Programming basics
(GKNB_INTA023)

Hatwagner F. Miklós, PhD.

Széchenyi István University, Győr, Hungary

https://github.com/sze-info/ProgrammingBasics
November 1, 2020

Searching for the longest 3D vectors

vector1.c

```c
#include <stdio.h>
#include <math.h>
#define MAX 1000

int main(void) {
  double x[MAX], y[MAX], z[MAX], length[MAX];
  int count;
  double maxLength = 0.;
  printf("Searching for the longest vectors\n"
         "Enter the number of vectors: ");
  scanf("%d", &count);
  for(int i=0; i<count; i++) {
    printf("X coordinate of vector %d: ", i+1); scanf("%lf", &x[i]);
    printf("Y coordinate: "); scanf("%lf", &y[i]);
    printf("Z coordinate: "); scanf("%lf", &z[i]);
    length[i] = sqrt(x[i]*x[i] + y[i]*y[i] + z[i]*z[i]);
    if(length[i] > maxLength) maxLength = length[i];
  }
```

Searching for the longest 3D vectors

vector1.c

```
  printf("Maximum length: %f, the longest vectors are:\n", maxLength);
  for(int i=0; i<count; i++) {
    if(length[i] == maxLength) {
      printf("%f %f %f\n", x[i], y[i], z[i]);
    }
  }
  return 0;
}
```

Problem:

- the X, Y, Z coordinates of a vector are more closely related than eg. the X coordinates of various vectors
- but our arrays do not reflect it

Structures

Main features:

- Easy handling of a group of logically related variables
- A compound, user-defined type can be created
- A group of one or more *members* with unique identifiers
- Possibilities:
  - Assignment (copy)
  - Can be passed to functions
  - Can be the return value of a function
- Impossible: comparison (possibly per member)
- *Almost* anything can become a member

Searching for the longest 3D vectors

vector2.c

```c
#include <stdio.h>
#include <math.h>
#define MAX 1000

struct vector {
  double x, y, z;
  double length;
};

int main(void) {
  struct vector av[MAX];
  int count;
  double maxLength = 0.;
  printf("Searching for the longest vectors\n"
         "Enter the number of vectors: ");
  scanf("%d", &count);
```

Searching for the longest 3D vectors

vector2.c

```c
  for(int i=0; i<count; i++) {
    printf("X coordinate of vector %d: ", i+1); scanf("%lf", &av[i].x);
    printf("Y coordinate: "); scanf("%lf", &av[i].y);
    printf("Z coordinate: "); scanf("%lf", &av[i].z);
    av[i].length = sqrt(
      av[i].x*av[i].x + av[i].y*av[i].y + av[i].z*av[i].z);
    if(av[i].length > maxLength) maxLength = av[i].length;
  }
  printf("Maximum length: %f, the longest vectors are:\n", maxLength);
  for(int i=0; i<count; i++) {
    if(av[i].length == maxLength) {
      printf("%f %f %f\n", av[i].x, av[i].y, av[i].z);
    }
  }
  return 0;
}
```

Structure declaration

General usage: **struct** *<structure-tag>*
<member-declarations> *<variable-declarations>*;

Example structure declaration

```
struct student { // Structure declaration
  char name[64];
  int pointsEarned;
};

struct student Jane, as[1000]; // Variable declarations
```

- `student` is the tag of the structure, it identifies the type together with keyword `struct`:
  `struct student Jane;`
- Members: `name, pointsEarned` (unique identiers (names) inside the structure)
- Variables: `Jane`
  `struct student as[1000];` an array of 1000 students

Structure declaration

Where should a structure be *declared*?

- In front of the first usage of the type
- Generally at the beginning of the source code, outside of all functions

All declarations create a *new and unique type* even if their members are the same

Different types

```
struct student1 {
  char name[64]; int pointsEarned;
};

struct student2 {
  char name[64]; int pointsEarned;
};

struct student1 Jane;
struct student2 Joe;
Jane = Joe; // error: incompatible types when assigning to type
            // 'struct student1' from type 'struct student2'
```

Where should a structure be *defined*? → In the narrowest possible scope

Structure member declaration

- A member can be eg.
  - an already declared structure
  - an embedded structure, even without tag
  - array
  - (a function pointer)
- The name of the member must be unique only inside the structure
- The semicolon (;) at the end of the declaration cannot be omitted!

Valid member declarations

```
struct s { int i; };
struct member_decl {
  struct s s1;
  struct { int i; long l; } e;
  int numbers[30];
};
```

Structure member declaration

A member's type cannot be eg.

- `void`
- itself
- function

Invalid member declarations

```
struct incomplete;
struct member_error {
  void v; /* error: variable or field ‘v’ declared void */
  struct incomplete s; /* error: field ‘s’ has incomplete type */
  struct member_error me; /* error: field ‘me’ has incomplete type */
};
```

Remark: an incomplete array (→ its size is unknown to the compiler) can be a member according to the C99 standard, if certain conditions are met.

Accessing structure members

Member access operator

- *structure.member*
- High precedence operator, the direction of associativity is from left to right

Accessing structure members, assignments

```
struct student {
  char name[64];
  char neptun[7];
  struct {
    int day, month, year;
  } birth;
};
/* ... */
struct student Jane;
strcpy(Jane.name, "Jane Doe");
strcpy(Jane.neptun, "A1B2C3");
Jane.birth.day = 2; Jane.birth.month = 1;
Jane.birth.year = 1990;
```

Initialization of structures

The members are initialized one after another to the values in the initializer list. A structure of the same type can also be az initializer.

Initialization of structures

```
struct student {
  char name[64], neptun[7];
  int day, month, year;
};

struct student Jane =
  { "Jane Doe", "A1B2C3", 23, 4, 1990 };
struct student Mary = Jane;
```

Initialization of structures

Initialization of embedded structures: with embedded initializers

Initialization of an embedded structure and array

```
struct date {
  int day, month, year;
};

struct student {
  char name[64], neptun[7];
  struct date birth, graduation;
};

struct student Jane = { "Jane Doe", "A1B2C3",
  {23, 4, 1990}, {3, 6, 2015} };
```

- The count of initializer list elements must not exceed the number of structure members!
- If it has fewer elements $\rightarrow$ all remaining bits are going to be set to zero
- In case of embedded types the `{ }` can be omitted or can be even placed around all initializers, but it is recommended to follow the internal structure of the type.

Initialization of structures

Usage of *designators*: direct references to the members (C99)

Initialization of an embedded structure and arrays with designators

```
struct student {
  char name[64], neptun[7];
  struct {
    int day, month, year;
  } birth;
} Jane = { .name="Jane Doe", .neptun="A1B2C3",
           {23, 4, 1990} };
```

In case of a missing designator the initialization continues with the member that follows the member referenced by a designator for the last time. The order of designator usage is arbitrary.

Date management

calendar1.c

```c
struct date {
  int day, month, year;
};

bool leap(int year) { // leap year detection
  return (year%4==0 and year%100!=0) or year%400==0;
}

int daysOfMonth(int year, int month) { // returns the days
  int ad[12] =                         // in a given month of a year
    { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
  if(month == 2) {
    if(leap(year)) return 29; else return 28;
  } else {
    return ad[month-1];
  }
}
```

Date management

calendar1.c

```c
bool check(struct date d) { // content validation
  if(d.month<1 or d.month>12) return false;
  int days = daysOfMonth(d.year, d.month);
  if(d.day<1 or d.day>days) return false;
  return true;
}

int dayOfYear(struct date d) { // determining the day of the year
  int days = d.day;             // based on year, month and day
  for(int month=1; month<d.month; month++) {
    days += daysOfMonth(d.year, month);
  }
  return days;
}
```

Date management

calendar1.c

```c
int base(struct date d) { // days elapsed since 01.01.0000
  int b = 0;
  for(int year=0; year<d.year; year++) {
    b += 365 + leap(year);
  }
  for(int month=1; month<d.month; month++) {
    b += daysOfMonth(d.year, month);
  }
  b += d.day;
  return b;
}

int difference(struct date begin, struct date end) { // days elapsed
  return base(end)-base(begin);           // between begin and end dates
}
```

Date management

calendar1.c

```c
// determining month and day based on the day of the year
struct date monthAndDay(int year, int dayOfYear) {
  struct date d = { dayOfYear, 1, year };
  int day;
  for(d.month=1;
      d.day>(day=daysOfMonth(year, d.month)); d.month++) {
    d.day -= day;
  }
  return d;
}
```

Date management

calendar1.c

```c
int main(void) {
  struct date d = {23, 10, 2020};
  printf("The given date is %s.\n"
         "%d.%d.%d is the %dth day of the year.\n",
         (check(d)?"valid":"invalid"), d.day, d.month, d.year,
         dayOfYear(d));
  struct date xmas = {24, 12, 2020};
  printf("How many days are left to christmas? %d\n",
    difference(d, xmas));
  int dy = 300;
  d = monthAndDay(d.year, dy);
  printf("The %dth day of %d is: %d.%d\n",
    dy, d.year, d.day, d.month);
  return 0;
}
```

Date management

Output

```
The given date is valid.
23.10.2020 is the 297th day of the year.
How many days are left to christmas? 62
The 300th day of 2020 is: 26.10
```

Drawing rectangles

Output (1/2)

```
Please enter the data of rectangles!
X coordinate of the top left corner of rectangle #1: [0, 78] (enter a negative value to exit) 1
Y coordinate of the top left corner rectangle #1 [0, 23] 1
X coordinate of the bottom right corner rectangle #1 [2, 79] 11
Y coordinate of the bottom right corner rectangle #1 [2, 24] 11
Drawing character of rectangle #1: |
X coordinate of the top left corner of rectangle #2: [0, 78] (enter a negative value to exit) 6
Y coordinate of the top left corner rectangle #2 [0, 23] 6
X coordinate of the bottom right corner rectangle #2 [7, 79] 16
Y coordinate of the bottom right corner rectangle #2 [7, 24] 16
Drawing character of rectangle #2: +
X coordinate of the top left corner of rectangle #3: [0, 78] (enter a negative value to exit) 15
Y coordinate of the top left corner rectangle #3 [0, 23] 2
X coordinate of the bottom right corner rectangle #3 [16, 79] 30
Y coordinate of the bottom right corner rectangle #3 [3, 24] 7
Drawing character of rectangle #3: -
X coordinate of the top left corner of rectangle #4: [0, 78] (enter a negative value to exit) -1
...
```

Drawing rectangles

Output (2/2)

```
...
 |||||||||||
 |||||||||||   ----------------
 |||||||||||   ----------------
 |||||||||||   ----------------
 |||||||||||   ----------------
 |||||+++++++++----------------
 |||||+++++++++----------------
 |||||+++++++++++
 |||||+++++++++++
 |||||+++++++++++
 |||||+++++++++++
      +++++++++++
      +++++++++++
      +++++++++++
      +++++++++++
      +++++++++++

...
```

Drawing rectangles

rectangle1.c

```c
#include <stdio.h>
#include <stdbool.h>
#include <iso646.h>
#define MAXSHAPE 128
#define MINX 0
#define MAXX 79
#define MINY 0
#define MAXY 24

struct coordinate {
  int x, y;
};
struct rectangle {
  struct coordinate tl, br; // top left, bottom right
  char c;                   // drawing character
};
```

Drawing rectangles

rectangle1.c

```c
int main(void) {
  struct rectangle ar[MAXSHAPE];
  int count=0, c; bool goon=true;
  printf("Please enter the data of rectangles!\n");
  while(count<MAXSHAPE and goon) {
    do {
      printf("X coordinate of the top left corner of rectangle #%d: "
             "[%d, %d] (enter a negative value to exit) ", count+1, MINX, MAXX-1);
      scanf("%d", &c);
      goon = c>=0;
    } while(goon && (c<MINX or c>MAXX-1));
    if(goon) {
      ar[count].tl.x = c;
      ar[count].tl.y = read(count+1, "Y coordinate of the top left corner", MINY, MAXY-1);
      ar[count].br.x = read(count+1, "X coordinate of the bottom right corner",
        ar[count].tl.x+1, MAXX);
      ar[count].br.y = read(count+1, "Y coordinate of the bottom right corner",
        ar[count].tl.y+1, MAXY);
      printf("Drawing character of rectangle #%d: ", count+1);
      scanf(" %c", &ar[count].c);
      count++;
    }
  }
  draw(ar, count);
  return 0;
}
```

Drawing rectangles

rectangle1.c

```c
int read(int count, char s[], int min, int max) {
  int k;
  do {
    printf("%s rectangle #%d [%d, %d] ",
      s, count, min, max);
    scanf("%d", &k);
  } while(k<min or k>max);
  return k;
}
```

Drawing rectangles

rectangle1.c

```c
bool isCovered(struct rectangle r, int row, int col) {
  return (r.tl.x<=col and r.br.x>=col) and
         (r.tl.y<=row and r.br.y>=row);
}

void draw(struct rectangle ar[MAXSHAPE], int count) {
  for(int r=MINY; r<=MAXY; r++) {
    for(int c=MINX; c<=MAXX; c++) {
      bool covered = false;
      for(int i=count-1; i>=0 and not covered; i--) {
        if(isCovered(ar[i], r, c)) {
          printf("%c", ar[i].c); covered = true;
        }
      }
      if(not covered) printf(" ");
    }
    printf("\n");
  }
}
```