Exercise 2

Programming III in C++

Computer Science

Content

Exercise 2.1

Implement a function `calcSurfaceAndVolumeOfSphere()` which has a return type `void` and expects three parameters: radius, surface and volume. It calculates the surface and volume of a sphere from the given radius. For a sphere, surface and volume are calculated according to following formulas:

- surface = 4π r * r
- Volume = (4 / 3)π r^3 (where π = 3.1415926536).

Implement this function in two variants:

a) with references

b) with pointers

Test these functions by calling the functions with the following values for the radius: to 0.5, 1, 1.5, to 10.0. Display the results within the calling function for surface and volume.

Exercise 2.2 – theoretical exercise

What values do the variables i1 and i2 have after calling func() in function main()?

```
int i1 = 1;
int i2 = 2;

void func(int & ref1, int & ref2) {
     int * ptr1;
     int * ptr2;
     ptr1 = &i1;
     ref2 = *ptr1;
     ptr2 = ptr1;
     *ptr2 = 4;
     ref1 = 5;
     ref2 = *ptr2;
}
int main(int argc, char* argv[])
{
     func(i1, i2);
     cout << "i1 = " << i1 << endl;
     cout << "i2 = " << i2 << endl;
     return 0;
}
```

Exercise 2.3 – theoretical exercise

The following declarations shall be given:

```
int v[] = {10, 20, 30, 40};
int i, *pv;
```

What will the following statements display on the screen?

```
a) for (pv = v; pv <= v + 3; pv++)
          cout << *pv << endl;

b) for (pv = v, i = 1; i <= 3; i++)
     cout << pv[i] << endl;

c) for (pv = v, i = 0; pv+i <= &v[3]; pv++, i++)
     cout << *(pv + i) << endl;
```

Note: Please do not program in this bad style!

Exercise 2.3

Enclosed with this exercise sheet you will find the files testSimpleList.cpp and list.h including the type definitions and functions for singly linked lists, which were introduced in the slide set “Recursive Data Structures”, as well as some test calls in the main function.

a) Develop a function

```
void printList(List lst);
```

which outputs all elements of a list (i.e., the value of the info component respectively) via cout.

b) Develop a function

```
void insertLast(List& lst, int info);
```

which creates a new list element with the content `info` and appends this new element as the last element to the end of the linked list `lst`.

c) Develop a function

```
void reverseList(List& lst);
```

which reverses the order of the elements of a singly linked list. For example, if the list previously contains elements with the values 1, 2, 3, 4, then after the call the list elements should be arranged in the order 4, 3, 2, 1.

This function should be implemented in such a way that no new list elements are created, but only the pointers of the existing elements are "relocated".

Tip:
Build the inverse list by starting with an empty list and then gradually "unhooking" the first element from the original list and "hanging" it in front of the opposite list.

Split your program into separate source files List.cpp, List.h and testSimpleList.cpp so that the complete list functionality is implemented in List.cpp and List.h.