

Mechatronic Engineering

Object Oriented Programing and Software Engineering
Laboratory instruction 11
C++ introduction

AGH Kraków, 2020

Materials created for educational purposes.
Dedicated for students attending Software Engineering course.
Author would appreciate any feedback regarding errors of any kind found in the instruction script.
Please report those to the following email address: daniel.t@agh.edu.pl

Spis treści

1	Structures	4
2	Unions	5
3	Bit fields	6

1 Structures

In C++, a structure is a class that, in principle, has all public elements. This is the opposite of a standard class whose elements are implicitly private. Unlike classical C, structures in C++ (like any class) can contain methods.

```
1 #include <iostream>
2 #include <string.h>
3
4 struct Books{
5     char title[50];
6     char author[50];
7     char subject[100];
8     int book_id;
9     void printBook();
10 };
11
12
13 int main(void){
14
15     Books Book1;      /* Declare Book1 of type Books */
16     Books Book2;      /* Declare Book2 of type Books */
17
18     /* book 1 specification */
19     strcpy( Book1.title, "Heaven's Devisl");
20     strcpy( Book1.author, "William C. Dietz");
21     strcpy( Book1.subject, "Starcraft novel");
22     Book1.book_id = 6495407;
23     /* book 2 specification */
24     strcpy( Book2.title, "Misery");
25     strcpy( Book2.author, "Stephen King");
26     strcpy( Book2.subject, "Psychological thriller");
27     Book2.book_id = 6495700;
28     /* print Book1 info */
29     Book1.printBook(); /* Print Book2 info */
30     Book2.printBook();
31
32     return 0;
33 }
34 void Books::printBook(){
35     std::cout << "Book title : " << title << "\n";
36     std::cout << "Book author : " << author << "\n";
37     std::cout << "Book subject : " << subject << "\n";
```

```
38     std::cout << "Book book_id : " << book_id << "\n";
39     std::cout << std::endl;
40 }
```

2 Unions

Union can be best described as a virtual data storage box. It allows you to store different types of objects in the same place in memory. Only one object can be stored at a time. Union is used to save memory. Its size is determined by the size of the largest of the objects to be stored inside it.

```
1 union box {
2     long long int bigObject;
3     char smallObject;
4     float numbObject;
5 };
```

In the example above, the union has three components of types: long long int, float and char. Each of them occupies 8, 4 and 1 bytes in memory. In this situation, the union will be stored as 8 byte box. If each of the elements were created as a separate variable, their declaration would reserve 13 bytes in the device's memory.

In unions, as in classes, you can use words that specify access (public, private). Due to the fact that unions do not support inheritance the word protected is not allowed.

It is not allowed to use a class object having a constructor or destructor as a union element.

If the union does not have a constructor, it can be initiated by usage of data corresponding to the type of the first union element.

```
1 union box {
2     long long int bigObject;
3     char smallObject;
4     float numbObject;
5 };
6
7 box powerLvl = {9000};
8 std::cout << "It's over " << powerLvl.bigObject << "\n";
```

The above example shows the initialization of the union object and how to refer to its components. As with other class objects, we refer to them after

the operator `.` and the operator `->` for the component shown by a pointer.

It is possible to create an anonymous union. It doesn't have its name. It cannot be followed by a declarator. An anonymous union is not a type, it defines an unnamed object and it cannot have member functions. If a specific object of the anonymous union is created, it ceases to be anonymous.

```
1 union {
2     int tmpNumber;
3     char tmpChar;
4     double tmpFP;
5 };
6
7 tmpFP = 3.14;
8 std::cout << "Pi = " << tmpFP <<std::endl;
9 tmpNumber = 42;
10 std::cout << "What is the Answer to the Ultimate Question of Life,
    The Universe, and Everything?\n" << tmpNumber;
```

When referring to the anonymous union component, there is no need to use the operator `'.'`. Components can be referred to as a regular variable. It should be remembered that the name of the component cannot be the same as the name of another variable operating in the same validity range.

3 Bit fields

Bit fields are special components of the class that allow you to specify the number of bits on which information will be stored.

```
1 class data {
2     int number :3;
3     unsigned int a :1,
4                 b :3,
5                 c :2;
6 };
```

The bit field can be of any integer type. Data from bit fields are placed in memory next to each other, the order determines the order of declarations. You can use unnamed fields to accurately arrange bit fields in memory. They are especially useful for manually placing individual fields in a word (a word is a group of bits, its size depends on the system architecture, the word for 8-bit systems is 8 bits long, 16-bit systems have a word length of 16 and so on). If the bit field can't fit in the word, it is moved completely to the next

one.

```
1 class data {
2     int number :3;
3     unsigned int  a :1,
4                   :2,
5                   b :3,
6                   c :2;
7 };
```

The following example shows how to use material presented in the manual in practice.

Example:

Source: *J. Grębosz, Symfonia C++ standard*

```
1 #include <iostream>
2
3 using namespace std;
4
5 const int max_urzadz = 15;
6 const int max_czujn = 10;
7
8
9 int detektor[max_urzadz][max_czujn]; // tablica na wyniki pomiarow
10
11 bool odczytaj_zdarzenie();
12 void analizuj_zdarzenie();
13 unsigned long pobierz_slowo();
14
15 int main(){
16     cout << "Program analizuje dane pomiarowe" << endl;
17
18     while (odczytaj_zdarzenie()){ //odebranie danych
19         analizuj_zdarzenie();
20     }
21     cout << "Nie ma wiecej danych" << endl;
22
23     return 0;
24 }
25
26 bool odczytaj_zdarzenie(){
27     struct slowo_vxi {
28         unsigned int dana :16;
```

```

29     unsigned int urzadzenie :8;
30     unsigned int czujnik :6;
31     unsigned int      :2;
32
33     };
34     //unia pozwalajaca na poszatkowanie slowa
35     union {
36         unsigned long cale;
37         slowo_vxi      vxi;
38     };
39
40     cout << "Wczytywanie nastepnego zdarzenia...\n";
41     //petla wczytujaca wiele slow do jednego zdarzenia
42     while(1){
43         cale=pobierz_slowo();
44         if(!cale) return false;
45         if (vxi.urzadzenie == 0xf8){
46             cout<<"Koniec odczytu danych zdarzenia nr"<<vxi.dana<<endl;
47             return true;
48         }
49         else {
50             if(vxi.urzadzenie > max_urzadz || vxi.czujnik >=
51                 max_czujn){
52                 continue;
53             }
54             detektor[vxi.urzadzenie][vxi.czujnik]=vxi.dana;
55         }
56     }
57
58     void analizuj_zdarzenie(){
59         cout<<"Analiza zdarzenia. Zadzialaly:"<<endl;
60
61         for (int u = 0; u < max_urzadz; u++) {
62             for (int c = 0; c < max_czujn; c++) {
63                 if(detektor[u][c]){
64                     cout<<"\turzadzenie " <<u<<"czujnik " <<c<<" , odczyt="
65                         <<detektor[u][c]<<endl;
66                 }
67             }
68         }
69     }

```



```
70 unsigned long pobierz_slowo(){
71     unsigned long slowo[]={
72         0x4060658, 0x60201ff, 0x9058c, 0xf80000,
73         328457, 100729404, 0xf80001,
74         197827, 134417127, 84087033, 50927293,
75         16848207, 17105686, 16847128,
76         0xf80002,
77         0};
78     static int licznik;
79     return slowo[licznik++];
80 }
```
