# Mechatronic Engineering

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

AGH Kraków, 2021

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: danielt@agh.edu.pl

# Table of contents

# 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.

---

```
#include <iostream>
#include <string.h>

struct Books{
   char title[50];
   char author[50];
   char subject[100];
   int book_id;
   void printBook();
};


int main(void){

   Books Book1;   /* Declare Book1 of type Books */
   Books Book2;   /* Declare Book2 of type Books */

   /* book 1 specification */
   strcpy( Book1.title, "Heaven's Devisl");
   strcpy( Book1.author, "William C. Dietz");
   strcpy( Book1.subject, "Starcraft novel");
   Book1.book_id = 6495407;
   /* book 2 specification */
   strcpy( Book2.title, "Misery");
   strcpy( Book2.author, "Stephen King");
   strcpy( Book2.subject, "Psychological thriller");
   Book2.book_id = 6495700;
   /* print Book1 info */
   Book1.printBook();
   /* Print Book2 info */
   Book2.printBook();

   return 0;
}
void Books::printBook(){
   std::cout << "Book title : " << title << "\n";
```

```
   std::cout << "Book author : " << author << "\n";
   std::cout << "Book subject : " << subject << "\n";
   std::cout << "Book book_id : " << book_id << "\n";
   std::cout << std::endl;
}
```

---

## 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.

---

```
union box {
   long long int bigObject;
   char smallObject;
   float numbObject;
};
```

---

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.

---

```
union box {
   long long int bigObject;
   char smallObject;
   float numbObject;
};
```

```
box powerLvl = {9000};
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.

---

```
union {
   int tmpNumber;
   char tmpChar;
   double tmpFP;
};

tmpFP = 3.14;
std::cout << "Pi = " << tmpFP <<std::endl;
tmpNumber = 42;
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.

---

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

```
};
```

---

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.

---

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

---

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

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

---

```
#include <iostream>

 using namespace std;

const int max_devic = 15;
const int max_sensr = 10;


int detector[max_devic][max_sensr]; // array for the
results of measurements

bool read_event();
void analyse_event();
unsigned long get_word();
```

```
int main(){
   cout << "This software analyses the measured data"
<< endl;

while (read_event()){ //obtaining the data
   analyse_event ();
}
   cout << "There is no more data" << endl;

   return 0;
}

bool read_event (){
   struct word_vxi {
        unsigned int data :16;
        unsigned int devic :8;
        unsigned int sensr :6;
        unsigned int   :2;

   };
//union, which allows shredding a word
union {
   unsigned long whole;
   word_vxi  vxi;
};

cout << "Loading next event...\n";
//Loop, which loads many words form one event
while(1){
   whole=get_word();
   if(!whole) return false;
   if (vxi.devic == 0xf8){
      cout<<"End     of     loading     events     data
   number"<<vxi.data<<endl;
      return true;
}
   else {
        if(vxi.devic  >  max_devic  ||  vxi.sensr  >=
      max_sensr){
             continue;
        }
   detector[vxi.devic][vxi.sensr]=vxi.data;
   }
```

```
   }
}

void analyse_event(){
   cout<<"Event analysis. The following worked:"<<endl;

   for (int u = 0; u < max_devic; u++) {
        for (int c = 0; c < max_sensr; c++) {
             if(detector[u][c]){
                  cout<<"\tdevice         "<<u<<"sensor
             "<<c<<", odczyt= "<<detector[u][c]<<endl;
             }
        }
   }
}

unsigned long get_word(){
   unsigned long word[]={
        0x4060658, 0x60201ff, 0x9058c, 0xf80000,
        328457, 100729404, 0xf80001,
        197827, 134417127, 84087033, 50927293,
        16848207, 17105686, 16847128,
        0xf80002,
        0};
   static int counter;
   return word[counter++];
}
```

---