Wstęp do programowania w C

Marek Piotrów - Wykład 13
Od C do C++. Wprowadzenie do C++

27 stycznia 2015

Krótka historia języka C++ - przypomnienie

- 1979 - pierwsza wersja języka C++ stworzona przez B. Stroustrup'a jako obiektowe rozszerzenie języka C (pod wpływem języka Simula);
  kompilator **Cfront** tłumaczył kod z C++ na C;
- 1985 - opis języka C++ przez B. Stroustrup'a;
- 1998 - opublikowanie standardu języka C++:
  ISO/IEC 14882:1998 (c++98);
- 2011 - nowy standard języka C++:
  ISO/IEC 14882:2011 (c++0x, c++11)

Podstawowe właściwości języka C++

- nie jest (podobnie jak C) własnością żadnej osoby czy instytucji;
- jest językiem wieloparadygmatowym (można w nim programować przede wszystkim obiektowo, ale też generycznie i proceduralnie);
- próbuje zachować jak największą zgodność na poziomie kodu źródłowego i łączenia modułów skompilowanych z językiem C;
- zakłada statyczną kontrolę typów (podobnie jak C);
- umożliwia programiście bezpośrednie zarządzanie pamięcią (podobnie jak C);
- zakłada, że wydajność programów napisanych w C++ powinna być porównywalna z odpowiednimi napisanymi w C.

Pierwszy przykład I

```
#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

#define losuj(n) (n < 1? 0 : rand()%(n))

inline void zamien (int &p, int &q)
{
   int i=p ;
   p=q, q=i ;
}

void permutacja (int rozm, int *tab)
{
   for (int i=0; i < rozm; i++)
      tab[i]=i;
   for (int j=7; j > 0; j--)
      for (int i=0; i < rozm; i++)
         zamien(tab[i],tab[losuj(i+1)]) ;
}

void drukuj (int rozm, int *tab)
{
   cout<<"Losowa permutacja: ";
   for (int i=0; i < rozm; i++)
```

Pierwszy przykład II

```
    cout<<(i > 0 ? ',' :'[')<<tab[i];
  cout<<']'<<endl;
}

int main(void)
{
  const int rozmiar=10 ;
  int tab[rozmiar] ;

  permutacja(rozmiar,tab);
  drukuj(rozmiar,tab);

  char nazwa[128];
  cout<<"podaj nazwe pliku: ";
  cin>>nazwa;

  ofstream plik;
  plik.open(nazwa,ios::out|ios::binary);
  plik<<rozmiar<<": ";
  for (int i=0; i < rozmiar; i++)
    plik<<tab[i]<<' ';
  plik<<endl;
  plik.close();
  return 0;
}
```

Czego nie wolno w C++ (a można w C)

1. Przypisywać bez rzutowania wskaźnika typu `void *` (na przykład wartości zwracanej przez malloc).
2. Używać nazwy funkcji bez jej wcześniejszej deklaracji lub definicji (standard c99 też tego wymaga, ale kompilator gcc generuje w takiej sytuacji ostrzeżenie, a nie błąd).
3. Używać tablic o zmiennej liczbie elementów, obliczeń na liczbach zespolonych, inicjalizatorów desygnowanych czy stałych strukturalnych (które wprowadził standard C99).

Nowe elementy w C++

1. Klasy i obiekty, dziedziczenie.
2. Mechanizmy hermetyzacji (sekcje prywatne, chronione i publiczne).
3. Obiektowe operacje wejścia/wyjścia.
4. Obsługa wyjątków;
5. Operatory `new` i `delete`.
6. Przestrzenie nazw i operator zakresu `::` .
7. Zmienne i parametry referencyjne.
8. Funkcje i operatory przeciążone.
9. Argumenty domniemane.
10. Domyślne definicje funkcji otwartych w klasach.

Przykład użycia klasy

```
# include <iostream>
# include <cstring>

using namespace std;

class kolejka
{
private:
   class element
   {
   friend class kolejka;
   private:
      char *nazwa;
      element *nastepny;
   public:
      element(element *&lista, const char *nazwa=NULL);
      ~element(void);
   public:
      const char * wartosc(void) { return(nazwa); }
   };
   element *lista;
public:
   kolejka(void);
   ~kolejka(void);
public:
   void wstaw(const char *nazwa);
   void usun(void);
   const char * podaj(void);
};
```

Konstruktory i destruktory

```
kolejka::element::element(element *&lista, const char *nazwa)
{
   this->nazwa = new char[(nazwa ? strlen(nazwa) : 0) + 1];
   strcpy(this->nazwa,nazwa ? nazwa : "");
   if (lista != NULL) {
      nastepny=lista->nastepny;
      lista->nastepny=this;
   }
   else nastepny=this;
   lista=this;
}

kolejka::element::~element(void)
{
   delete nazwa;
}

kolejka::kolejka(void)
{
   lista=NULL;
}

kolejka::~kolejka(void)
{
   while (lista != NULL) usun();
}
```

Kolejka - wstawianie i usuwanie

```
void kolejka::wstaw(const char *nazwa)
{
   new element(lista,nazwa);
}

void kolejka::usun(void)
{
   if (lista == NULL)
      return;
   if (lista->nastepny == lista)
   {
      delete lista;
      lista=NULL;
      return;
   }
   element *pom;
   pom=lista->nastepny;
   lista->nastepny=pom->nastepny;
   delete pom;
}

const char *kolejka::podaj(void)
{
   return (lista ? lista->nastepny->wartosc() : NULL);
}
```

Kolejka - funkcja `main`

```
int main(void)
{
   char buf[128];
   kolejka kol;

   cout<<"==========="<<endl;
   for (int i=0; i < 10; i++) {
      cin.getline(buf,sizeof(buf));
      kol.wstaw(buf);
   }
   cout<<"==========="<<endl;
   for (int i=0;i < 10; i++) {
      cout<<kol.podaj()<<endl;
      kol.usun();
   }
   cout<<"==========="<<endl;
   return 0;
}
```

Deklaracja klasy `tekst`

```
#include <iostream>
#include <iomanip>
#include <cstring>

using namespace std;

class tekst
{
private:
   int dlugosc;
   char *tekstp;
private:
   void kopiuj(const tekst &t) {
      if (t.tekstp == NULL) {
         dlugosc=0; tekstp=NULL;
      }
      else {
         dlugosc=t.dlugosc;
         tekstp=new char[dlugosc+1];
         strcpy(tekstp,t.tekstp);
      }
   }
```

Konstruktory i destruktory

```
public:
   tekst(): dlugosc(0),tekstp(NULL) { }
   tekst(const tekst &t) {
      kopiuj(t);
   }
   tekst(const char *tp) {
      if (tp == NULL) {
         dlugosc=0; tekstp=NULL;
      }
      else {
         dlugosc=strlen(tp);
         tekstp=new char[dlugosc+1];
         strcpy(tekstp,tp);
      }
   }
   tekst(int n,char c=' ') {
      dlugosc=n;
      tekstp=new char[dlugosc+1];
      memset(tekstp,c,dlugosc);
      tekstp[dlugosc]='\0';
   }
   ~tekst() {
      if (tekstp != NULL)
         delete tekstp;
   }
```

Operacja podstawienia, dodawania i we/wy

```
public:
   tekst & operator=(const tekst &t) {
      if (this == &t) return *this;
      if (tekstp != NULL) delete tekstp;
      kopiuj(t);
      return *this;
   }
   const tekst operator+(const tekst &t) const;
   friend istream & operator>>(istream &os,tekst &t);
   friend ostream & operator<<(ostream &os,const tekst &t);
};
```

Operacja dodawania

```
const tekst tekst::operator+(const tekst &t) const
{
  tekst wynik;

  if (t.tekstp == NULL)
    wynik=*this;
  else if (this->tekstp == NULL)
    wynik=t;
  else {
    wynik.dlugosc=this->dlugosc+t.dlugosc;
    wynik.tekstp=new char[wynik.dlugosc+1];
    strcpy(wynik.tekstp,this->tekstp);
    strcpy(wynik.tekstp+this->dlugosc,t.tekstp);
  }
  return wynik;
}
```

Operacje we/wy

```
istream & operator>>(istream &is,tekst &t)
{
   char buf[200];
   is.getline(buf,sizeof(buf));
   if (t.tekstp != NULL) delete t.tekstp;
   t.dlugosc=strlen(buf);
   t.tekstp=new char[t.dlugosc+1];
   strcpy(t.tekstp,buf);
   return is;
}

ostream & operator<<(ostream &os,const tekst &t)
{
   os<<setw(t.dlugosc)<<t.tekstp;
   return os;
}

int main(void)
{
   tekst t1(10,'a');
   tekst t2(t1);
   tekst t3,t4;

   t3=t2;
   cout<<t3<<endl;
   cin>>t4;
   cout<<t4<<endl<<t4+(t3+t4)<<endl;
   return 0;
}
```

Pola i metody statyczne

```
#include <iostream>

using namespace std;

class Figura2D {
protected:
   static const double maxX=2000.0,maxY=1000.0;
   static unsigned liczbaFigur;
   double x,y;
public:
   Figura2D(double x0=0.0,double y0=0.0) {
      if (x0 >= 0.0 && x0 <= maxX && y0 >= 0.0 && y0 <= maxY) {
         x=x0; y=y0;
      } else {
         x=y=0.0;
         cerr<<"Figura2D: inicjalizacja poza obszarem"<<endl;
      }
      ++liczbaFigur;
   }
   virtual ~Figura2D() { --liczbaFigur; }
   void przesun (double dx,double dy) { x+=dx; y+=dy; }
   static unsigned ileFigur() { return liczbaFigur; }
};

unsigned Figura2D::liczbaFigur = 0;
```

Klasy bazowe i pochodne

```
class Figura2D {
protected:
   double x,y;
public:
   Figura2D(double x0=0.0,double y0=0.0): x(x0),y(y0) { }
   void przesun (double dx,double dy) { x+=dx; y+=dy; }
};

class Prostokat: public Figura2D {
protected:
   double dl_x,dl_y;
public:
   Prostokat(double x,double y,double dx,double dy):
      Figura2D(x,y),dl_x(dx),dl_y(dy) { }
};

class Kolo: public Figura2D {
protected:
   double r;
public:
   Kolo(double x,double y,double r0): Figura2D(x,y),r(r0) { }
};

class Pierscien: private Kolo {
protected:
   double rw; // promien wewnetrzny
public:
   Pierscien(double x,double y,double r,double rw0): Kolo(x,y,r),rw(rw0) { }
};
```

Funkcje wirtualne

```
#include <iostream>

class Figura2D {
protected:
   double x,y;
public:
   Figura2D(double x0=0.0,double y0=0.0): x(x0),y(y0) { }
   void przesun (double dx,double dy) { x+=dx; y+=dy; }
   virtual void skaluj(double wsp=1.0) {}
   virtual void wypisz() { std::cout<<"Poz=("<<x<<','<<y<<')';}
};

class Kolo: public Figura2D {
protected:
   double r;
public:
   Kolo(double x,double y,double r0): Figura2D(x,y),r(r0) { }
   virtual void skaluj(double wsp=1.0) { r*=wsp; }
   virtual void wypisz() { Figura2D::wypisz(); std::cout<<",promien="<<r; }
};

class Pierscien: public Kolo {
protected:
   double rw; // promien wewnetrzny
public:
   Pierscien(double x,double y,double r,double rw0): Kolo(x,y,r),rw(rw0) { }
   virtual void skaluj(double wsp=1.0) { r*=wsp; rw*=wsp; }
   void przesun(double dx,double dy) { Figura2D::przesun(dx,dy); }
   virtual void wypisz() { Kolo::wypisz(); std::cout<<",promien wew.="<<rw; }
};
```