Структуры, объединения, перечисления

Определение

Структура в языке C++ - производный тип данных, который представляет какую-то определенную сущность, позволяет комбинировать элементы данных разных типов, в том числе и других структурных типов.

Структуры полезны, когда необходимо объединить несколько переменных с разными типами под одним именем или когда необходимо сгруппировать некоторые данные, например, контакт из книги адресов:

Контакт {имя, адрес, телефон}

Тогда применение структур упрощает решение задачи сортировки по двум параметром, например:

Вывести записи о контактах из книги адресов в алфавитном порядке имен, при наличии нескольких записей с одним именем – в порядке возрастания номеров телефонов

Синтаксис описания структуры

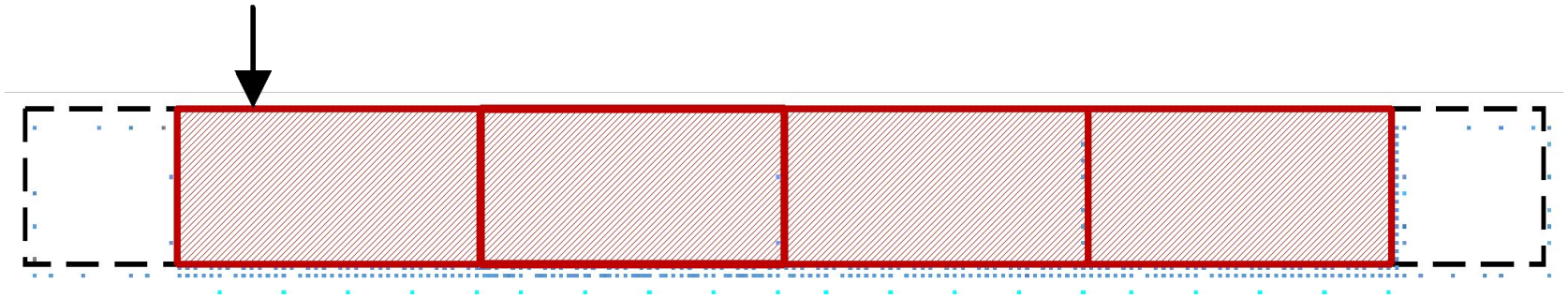
```
struct имя{
//список полей структуры
тип имя1;
тип имя2;
…
тип имяn;
} список_объектов;


struct имя{
//список полей структуры
…
}; //список может отсутствовать
```

Правила описания полей структуры

- поля могут иметь любой тип, кроме void, и типа самой структуры;
- поля могут являться указателем на саму структуру;

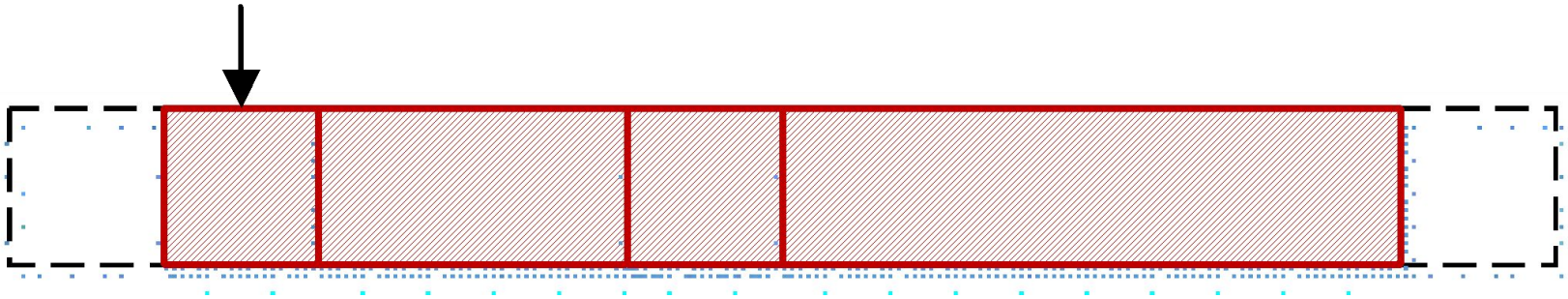
Представление массива в памяти

```
short sh_arr[4];
```



Представление структурного объекта в памяти

```
struct Mystruct{
bool flag;
short sh_var;
char ch;
float fl_var;
} str_obj;
```



*возможно выравнивание адресов в зависимости от настроек оптимизации компилятора

Пример описания структуры

```
struct Student{
unsigned short course;
char firstName[10];//внимание! В данной структуре
char lastName[10]; //используются разные представления строк
std::string speciality;//на практике так делать не рекомендуется
bool paymentForm;
};
```

Создание объекта структуры

```
//1 – C, C++ - при описании структурного типа
struct Student{/*описание полей структуры*/} Sidorov;

//2 – C++ - после создания структурного типа
Student Ivanov;

//3 – C, C++ - после создания структурного типа
struct Student Kozlov;

//4 – C++ (агрегатная инициализация)
Student Petrov = {2, "Petr", "Petrov", "MOAIS", true};

//5 – C++11+ (универсальная инициализация)
Student Volkov {3, "Vasia", "Volkov", "IVT", true};
```

*в языке C при определении структурных переменных необходимо применять ключевое слово struct

Создание безымянной структуры

```
struct {
//поля структуры
}obj;   /*obj – единственный объект структуры*/

struct Person{
string name;
size_t age;
struct {
   size_t width,
          height;
   } volume;
};
```

Вложенные структуры

```
struct Session2Chem{
   bool GeometryAndT;
   bool MathAanalysis ;
   unsigned DiscreteMath;
   unsigned AlgebraAndNT;
};

struct Student{
   unsigned course;
   char firstName[10];
   char lastName[10];
   std::string speciality;
   bool paymentForm;
   Session2Chem marks;
};
```

typedef и структуры

```
typedef struct{
//поля структуры
}MyStruct;   /* Mystruct – синоним структуры*/
```

Доступ к элементам структурированного объекта

```
Sidorov.course = 2;
strcpy(Sidorov.firstName, "Sidr");
strcpy(Sidorov.lastName, "Sidorov");
Sidorov.speciality = "MOAIS";
Sidorov.paymentForm = false;
```

Указатель на структурированный объект и доступ к полям через указатель

```
Student *SidorovsPtr;
SidorovsPtr = &Sidorov;
SidirovsPtr->PaymentForm = true;
/* аналогично
(*SidorovsPtr).PaymentForm = true;
*/
```

Массив элементов структурированного типа

```
const int grSize = 30;
Student stGroup[grSize];
for(int i=0; i < grSize; i++)
   cin >> stGroup[i].course
       >> stGroup[i].firstName
       >> stGroup[i].lastName
       >> stGroup[i].speciality
       >> stGroup[i].paymentForm;
```

Применение sizeof() к структуре и объектам структуры

```
sizeof (Student);  //32
sizeof (Sidorov);   //32
sizeof (SidorovsPtr);  //4
```

Создание объекта с вложенной структурой

```
Sidorov = {1, "Sidr", "Sidorov", "MOAIS", true,
                {0, true, 3, 4} };

if (Sidorov.marks.GeometryAndT = true)
   strcpy(Sidorov.firstName, "Very glad");
```

Объявление структуры

```
struct MyPoint;


struct Pixel{
std::string color;
MyPoint * point; /* нельзя создавать объекты неопределенной
  структуры, но можно указатели*/
};
…
struct MyPoint{
int x;
int y;
};
```

Указание значений в перечислении

enum имя {список_констант}; // not larger than int

Стандартный ввод и вывод ничего не знают о «новом» типе, поэтому по умолчанию могут работать только с его целочисленными эквивалентами

```
enum Marks {two = 2, three = 4, four/*=5*/, five = 5, unknown};

int main(){
Marks mark;
int i; cin >> i;
switch (i){
        case 2: mark = two; break;
        case 3: mark = three; break;
        case 4: mark = four; break;
        case 5: mark = five; break;
        default : mark = unknown;
        }
//вывод mark – значения 2-6
}
```

Вывод значения перечисления в строковом виде 1

```
enum Marks {two = 2, three, four, five = 5, unknown = 6};

int main(){
Marks mark;
string str;
switch (mark){
        case two  : str = "two"; break;
        case three: str = "three"; break;
        case four : str = "four"; break;
        case five : str = "five"; break;
        default   : str = "unknown";
        }
cout << str;
}
```

Вывод значения перечисления в строковом виде 2

```
enum Marks {two = 2, three, four, five, unknown};

int main(){
string sMarks[7] = {"unknown", "unknown", "two", "three",
          "four", "five", "unknown"};
int i = 3;
Marks mark = four;
cout << sMarks[i];
cout << sMarks[mark];
}
```

Перечисления. Операции и функции

Над объектами перечислений можно выполнять арифметические операции и операции сравнения

```
enum Marks{two = 2, three, four, five, unknown};
int main(){
Marks mark1 = three, mark2 = five;
int dec = abs(mark2 - mark1);
if (mark1 < mark2)
 cout << "Your second mark is greater";
if (mark1 == two || mark1 == tree)
 cout << "not receive a scholarship";
else
 cout << "receive a scholarship";
```

Текущий контроль

Какие из высказываний истинны?

а) имя поля структуры может совпадать с именем самой структуры

б) все имена полей структуры должны быть различны

в) все типы полей структуры должны быть различны

г) структура может содержать одно поле

Текущий контроль

Какие из высказываний истинны?

а) имя поля структуры может совпадать с именем самой структуры

б) все имена полей структуры должны быть различны

в) все типы полей структуры должны быть различны

г) структура может содержать одно поле

Текущий контроль

Какое количество байт должен занимать объект описанной ниже структуры

```
struct Time{
      unsigned long int hour;
      unsigned short int minute;
      unsigned short int second;
      };
```

а) 3
б) 4
в) 6
г) 8
д) 12

Текущий контроль

Какое количество байт должен занимать объект описанной ниже структуры

```
struct Time{
      unsigned long int hour;
      unsigned short int minute;
      unsigned short int second;
      };
```

а) 3
б) 4
в) 6
г) 8
д) 12

Текущий контроль

Какое значение будет храниться в переменной diff после выполнения программы?

```
struct NewStruct{
string name;
int count;
float cost;
};

int main(){
NewStruct someObj1{"Pen", 243, 17.98};
NewStruct someObj2 {"Eraser", 109, 10.72};
if (someObj1.name < someObj2.name){
	someObj1.count -= 10;
	someObj2.count+=10;
	}
else{
	someObj2.count -= 10;
	someObj1.count+=10;
	}
int diff = someObj1.count - someObj2.count;
```

Текущий контроль

Какое значение будет храниться в переменной diff после выполнения программы?

```
struct NewStruct{
string name;
int count;
float cost;
};

int main(){
NewStruct someObj1{"Pen", 243, 17.98};
NewStruct someObj2 {"Eraser", 109, 10.72};
if (someObj1.name < someObj2.name){
	someObj1.count -= 10;
	someObj2.count+=10;
	}
else{
	someObj2.count -= 10;
	someObj1.count+=10;
	}
int diff = someObj1.count - someObj2.count;    //154
```

Текущий контроль

В каком(их) из вариантов описания структуры, написанном на языке C++11+, не содержится синтаксических ошибок?

а)
```
struct a{
int b, c;
float d;}
```

б)
```
struct {
int b;
int c;
float d;}e;
```

в)
```
typedef struct {
int b, c;
float d;} e;
```

г)
```
struct a{
int b = 0;
int c;
float d;};
```

Текущий контроль

В каком(их) из вариантов описания структуры, написанном на языке C++11+, не содержится синтаксических ошибок?

```
а) struct a{
int b, c;
float d;}
```

```
б) struct {
int b;
int c;
float d;}e;
```

```
в) typedef struct {
int b, c;
float d;} e;
```

```
г) struct a{
int b = 0;
int c;
float d;};
```

Пример использования перечислений

```
enum Color {red, orange, yellow, green, skyBlue, blue,
margenta};
int main(){
Color x;
   /*must write
   /enum Color x;
   /for programming in C */
switch (x) {
   case red: case orange: case yellow:
      cout << " warm"; break;
   case green: case skyBlue: case blue: case margenta:
      cout << "cold"; break;
   default: cout << "no such color";
   }
}
```

Еще пример использования перечислений

```
enum Suit  {hearts = 3, dimonds, cross, peaks};
enum Rank  {six = '6', seven, eight, nine, ten = '1', jack = 'J',
      queen = 'Q', king = 'K', ace = 'A'};

struct Card{
Suit suit;
Rank rank;
};

int main(){
Card card;
int s; char r;
cin >> r >> s;
card.suit = (Suit)s; card.rank = (Rank)r;
cout << (char)card.rank << " "
     << (char)card.suit << endl;
}
```

Пример: структура + перечисление

```
enum Suit {/*…*/};
enum Rank {/*…*/};
struct Card{
/*…*/
};

int main(){//card2 ходит 1-м
Card card1, card2;
Suit trump;//козырь
if(card1.suit == card2.suit)
   if(card1.rank > card2.rank)
      cout << "1-st player win";
   else
      cout << "2-nd player win";
else
   if(card1.suit == trump)
      cout << "1-st player win";
   else
      cout << "2-nd player win";
}
```

Описание конструктора для структуры 1

Конструктор – функция, содержащая инструкции, которые необходимо выполнить при создании объекта структуры (или класса)

```
enum Mark {noSat = 2, sat, good, exc};
struct Session2Chem{
  bool GeometryAndT;
  bool MathAanalysis;
  Mark DiscreteMath;
  Mark AlgebraAndNT;
  Session2Chem(bool = 0, bool = 0, Mark = noSat,
           Mark = noSat);
};

Session2Chem::Session2Chem(bool t1, bool t2, Mark ex1, Mark ex2){
GeometryAndT  = t1;
MathAanalysis = t2;
AlgebraAndNT  = ex1;
DiscreteMath  = ex2;
}
```

Описание конструктора для структуры 2

```
enum Mark {noSat = 2, sat, good, exc};
struct Session2Chem{
  bool GeometryAndT;
  bool MathAanalysis;
  Mark DiscreteMath;
  Mark AlgebraAndNT;
  Session2Chem(bool = 0, bool = 0, Mark = noSat,
           Mark = noSat);
};


Session2Chem::Session2Chem(bool t1, bool t2, Mark ex1, Mark ex2) :
GeometryAndT (t1), MathAanalysis (t2), AlgebraAndNT  (ex1),
  DiscreteMath  (ex2) {}
```

Создание структурного объекта с использованием конструктора

```
Session2 rez1, //false, false, 2, 2
rez2 = Session2(true, true, exc, exc);
rez1 = rez2;
```

Пример конструктора в структуре

```
struct DynIntArray{
  size_t size;
  int * data;
  DynIntArray();
  DynIntArray(size_t);
  DynIntArray(size_t, int* dataSource);
  };

DynIntArray::DynIntArray(){
  size = 0;
  data = nullptr;
  }

  DynIntArray::DynIntArray(size_t newSize){
  size = newSize;
  data = new int [newSize]{0};
  }

  DynIntArray::DynIntArray(size_t, int* dataSource){
  ...
  }
```

Структурный объект в качестве аргумента функции

```
bool getTestSuccess(Student){
/*some algorithm to get test, return true or false*/
}

void retakeGeometry(Student st){//justForCopy
if (getTestSuccess(st))
  st.marks.GeometryAndT = true;
}

…
int main(){
Student Smirnov;
if (!Smirnov.marks.GeometryAndT)
  retakeGeometry(Smirnov);
}
```

Ссылка на структурный объект в качестве аргумента функции

```
bool getTestSuccess(Student&);
…
void retakeGeometry(Student &st){
if (getTestSuccess(st))
st.marks.GeometryAndT = true;
}

int main(){
Student Smirnov,
      *Sidorov = new Student;
if (!Smirnov.marks.GeometryAndT)
 retakeGeometry(Smirnov);
if (!Popov->marks.GeometryAndT)
 retakeGeometry(*Sidorov);
}
```

Указатель на структурный объект в качестве аргумента функции

```
bool getTestSuccess(Student*);
…
void retakeGeometry(Student *st){
if (getTestSuccess(st))
st->marks.GeometryAndT = true;
}

int main(){
Student Smirnov,
*Sidorov = new Student;
if (!Smirnov.marks.GeometryAndT)
 retakeGeometry(&Smirnov);
if (!Sidorov->marks.GeometryAndT)
 retakeGeometry(Sidorov);
}
```

Статический массив, хранящий структурированные данные

```
bool getTestSuccess(Student);
void retakeGeometry(Student);
…
int main(){
const unsigned int n = 25;
Student stds[n];
…//заполнение массива данными
for (int i = 0; i < n; i++)
  //if (!stds[i].marks.GeometryAndT)
    retakeGeometry(stds[i]);
}
```

Динамический массив, хранящий структурированные данные

```
…
bool getTestSuccess(Student*);
void retakeGeometry(Student*);
…

int main(){
unsigned int n;
//ввод n;
Student *stds = new Student[n];
…//заполнение массива данными
for (int i = 0; i < n; i++)
  //if (!stds[i].marks.GeometryAndT)
    retakeGeometry(&stds[i]); // stds + i
}
```

Возврат функцией структурированного объекта или указателя

```
Student findFirstDntGetGeomStud(
   Student *group, unsigned groupSize){
for (unsigned i = 0; i < groupSize; i++)
 if (!group[i].marks.GeometryAndT)
    return group[i];
}

Session2* SessionRezOfFirstDntGetGeomStud(
   Student *group, unsigned groupSize){
for (unsigned i = 0; i < groupSize; i++)
 if (!group[i].marks.GeometryAndT)
    return &(group[i].marks);
}
```

Описать функцию *secToTime* (seconds), определяющую по времени *seconds* (в секундах) содержащееся в нем количество часов, минут и секунд (*Time)*

```
struct Time{
unsigned int h, m, s;
Time (unsigned int h, unsigned int m, unsigned int s) :
     h(h), m(m), s(s){}
};

Time secToTime(unsigned int sec){
/*Time time;
time.s = sec % 60;
time.m = (sec / 60) % 60;
time.h = sec / 3600;
return time;*/
return Time (sec / 3600 , (sec / 60) % 60, sec % 60);
}
```

Битовые поля в структурах

Объявление_поля_структуры : константное_выражение;

```
struct data{
unsigned year  : 11;
unsigned month : 4;
unsigned day   : 5;
};
struct device{
unsigned active : 1;
unsigned ready  : 1;
unsigned error  : 2;
unsigned        : 3;
unsigned end    : 1;
};
```

Объединения

```
union имя{
тип имя1;
тип имя2;
…
тип имяn;
}список_объектов;
```

Пример объединения

```
enum Paytype {CARD, CASH};
struct Payment{
  Paytype ptype;
  union {
    char card[25];
    long cash;
  }info;
};

int main(){
Payment pay;
switch (pay.ptype){
  case CARD:
   cout << "Оплата по карте: " << pay.info.card; break;
  case CASH:
   cout << "Оплата наличными: " << pay.info.cash; break;
  }
}
```